Escuela Politécnica Superior

Object detection and tracking using an UAV

Bachelor's Degree in Computer Engineering

Bachelor's Thesis

Author:
Yerai Pastor Quiles

Supervisor:
Fidel Aznar Gregori

November 2018

Universitat d'Alacant
Universidad de Alicante

UNIVERSITY OF ALICANTE

BACHELOR'S THESIS

Object detection and tracking using an UAV

Author
Yerai PASTOR-QUILES

Advisor
Fidel AZNAR-GREGORI

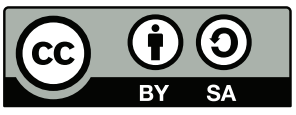
A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Bachelor's Degree in Computer Engineering
Department of Artificial Intelligence and Computing

14th November 2018

This document was proudly made with LaTeX.



'Well, if it can be thought, it can be done, a problem can be overcome'

E.A. Bucchianeri

Abstract

This project is enclosed in the field of computer **artificial vision** and developed by taking profit of the obtained knowledge during the **Computer Engineering degree** and the **computation** specialization.

Throughout the document we will scrutinize the **state of art** of the ongoing computational systems, analyzing the **history and evolution** of them. Owing to develop the final application designed to run in a **reduced size and performance** system, we will focus on several optimized computational methods, including *ad-hoc* developed Neural Networks models to serve for this purpose.

As a consequence of this analysis, we will develop an application capable of **recognizing objects** in real time, by using the video streaming from an **Unmanned Aerial Vehicle** (UAV) which will be running entirely on a medium budget smartphone.

As a result of this study, the developed application will be able to **automatically command** the aircraft to follow a determined target by using the recognitions in the video thus adapting to the position changes of the desired objective.

Resumen

Este proyecto se enmarca en el ámbito de la **visión artificial** por ordenador, haciendo uso de los conocimientos obtenidos durante el transcurso del grado de **Ingeniería Informática** y la especialización en la rama de **Computación**.

A lo largo del documento examinaremos el **estado del arte** de los sistemas computacionales actuales, analizando su **historia y evolución**. Con la finalidad de desarrollar una aplicación diseñada para ejecutarse en un entorno de **bajos recursos** computacionales, centraremos el objeto de estudio en algunos métodos de computación optimizados, incluyendo modelos de Redes Neuronales que han sido concebidos para cumplir con este propósito. Como consecuencia de este análisis, se llevará a cabo el desarrollo de una aplicación capaz de **reconocer objetos** en tiempo real, haciendo uso para ello, un enlace de vídeo obtenido desde un **vehículo aéreo no tripulado** (UAV) y se ejecutará en un Smartphone Android de gama media.

Como resultado de este estudio, la aplicación desarrollada será capaz de **comandar** de manera **autónoma** el movimiento de una aeronave con el propósito de que se persiga un determinado objetivo; este objetivo vendrá determinado por el algoritmo de reconocimiento, con lo que se actualizará el comando a enviar dependiendo de los cambios de posición del mismo.

Table of contents

List of Figures

Acronyms and abbreviations

- **UAV/Drone:** Unmanned Aerial Vehicle
- **Occusync:** Wireless data transmission protocol invented by DJI.
- **USB:** Universal Serial Bus. Communication and data transfer protocol.
- **VR:** Virtual Reality.
- **Android:** Operating system for mobile devices.
- **SDK:** Software Development Kit, provides the needed tools to develop applications.
- **FLOPS:** Floating-point operations per second.
- **CUDA:** Computed Unified Device Architecture. Parallel computing platform.
- **YUV:** Color space that use chrominance differences.
- **RGB:** Color space which acronym represent the primary colors Red Green and Blue.
- **ANN:** Artificial Neural Network.
- **CNN:** Convolutional Neural Network.
- **SSD:** Single Shot Multibox Detector. Neural network model used in the application.
- **YOLO** You Only Look Once. Neural Network analyzed in this project.
- **UML:** Unified Modeling Language, it is used to model the architecture and behaviour of a system.
- **IDE:** Integrated development environment. Provides some extra tools to a language.
- **Tensorflow:** Development library used in machine learning and deep learning.
- **XML:** Extensible Markup Language, used in the view creation of Android.
- **CSV:** Comma separated values, used to generate dataset information.
- **ROI:** Region of interest.
- **API:** Application programming interface. Provides some communication rules between applications.

- **H264:** Used transport codec in video transmitting.
- **CPU:** Central Procesing Unit.
- **GPU:** Graphíc Procesing Unit.

Introduction

This document contains all the information related with the development of the Computer Engineering Bachelor's Thesis.

The project was developed with the objective of studying the state of art of the modern computing algorithms centered in the artificial Vision field. We will cover this investigation of the most recent technologies related with this field in the first chapter of the project.

Following with the second chapter, we will introduce the concept of Neural Networks and explore some of the models that we are going to be using and referencing along the document.

The aim of the research process is creating a software system capable of operating in real time on a low resources device, such as a mobile device, with this purpose, the investigation process will be enriched by technologies whose characteristics might be profitable on these devices.

In the third chapter, will cover the implementation of the software and we will take advantage of the concepts embodied on both first and second chapters.

The developed software will comply with the following key features:

- Real time video reception from the camera of a determined Unmanned Aerial Vehicle (UAV), showing the image on a mobile device.
- Neural Network model trained to recognize a set of UAVs models. The selected model will be capable of running on the mobile device at the same time as the video reception, and execute the Neural Network mode over the received real time video without excessively compromising the usability of it.
- Autonomous movement of the UAV following an objective, which will be provided by the detections obtained from the execution of the Neural Network and will be commanded by the same mobile device in which we are executing the application.

Motivation and objectives

This project contains two fundamental parts which represent the motivation of engaging the development process:

Firstly, the development of this kind of application means a challenge, on account of the final goal to be a software system adequate to efficiently run on a mobile device, which implies that the used computational model should be highly optimized to prioritize the processing speed without excessively sacrificing the detection quality.

In the second place, the software to be developed should be capable of running in real time and will only need a mobile device (smartphone or tablet) and a drone, on the same way that it is currently being used in the market, without adding some extra components that difficult its usability.

The project will be divided in the following sections, where a series of objectives will be accomplished:

- **Video reception:** We will define the manner of obtaining the video link from the UAV, in addition to the way of showing the images on the mobile device screen. The video receiving will be running in parallel from the video processing, in order to not slowing the video shown to the user.

- **Video processing:** In this part of the project we will describe the steps taken to perform the processing on the video from the aircraft. This process consists on the execution of a Neural Network targetting for analyzing the image and identifying any known drone model.

- **Aircraft command:** By using the detections obtained from the execution of the Neural Network, the program will be qualified to command the aircraft to follow a specific detected object and sending update commands depending on the object position changes.

- **Result analysis:** The last stage on the project, in which all the data collected resulting from the implementation of the project and the junction of all its parts, will be analyzed and discussed. In this last section, comparatives of the studied methods will be included, and several conclusions will be extracted that will allow a synthesis of the work that has been carried out

As mentioned before, the features of receiving and decoding the video streaming from the drone, the processing of the images to recognize objects and the commanding of the drone will all be running simultaneously on the same mobile device, and that is where the biggest complexity of the system lies.

Chapter 1

Theoretical concepts and investigation.

This first chapter will serve to introduce some important theoretical concepts that will be used in the next chapters.

We will start by introducing the work methodology followed and analyzing the main categories, the iterative and agile models.

To conclude with this chapter, we will explain the concept of video processing in order recognize objects in an image by stating a series of artificial vision concepts related with the Artificial Vision field.

1.1 Work methodology

When the project started, the only methodology followed was a piece of paper with all the task needed to be done. After some weeks there was a need to switch the work methodology in order to provide some organization to the developing process.

The first research was related with the two main methodologies (1) classification: Iterative and Agile models.

- **Iterative model:** It is based on iterations where each iteration consists on a set of activities such as requirements analysis, design, programming and test. The main goal of each iteration is to generate a release which should be stable and partially completed. One key feature of each iteration is the fact that there can be no change of the work to be done once the iteration has started.

- **Agile model:** It is characterized by short iterations with adaptive plans. This provide rapid and flexible response to the changes in the requirements of the project. One key feature is the promotion of simplicity, lightness and programming over documenting.

The project goals were clear, but the way of implementing and achieving them was unknown, due to the fact that I had never work on this kind of systems and the technology used was newfangled. The need of researching the state of the art of these technologies was a key factor in order to choose an agile methodology since the requirements may vary and the work plan could change depending on the research done. A research about the 3 main agile methodologies was made in order to decide which one was more suitable for the project:

- **Scrum:** Scrum (2) emphasizes in self-organizing teams, daily team measurement and daily meetings. The work is divided into actions which then are including into an iteration called sprint. The duration of the iterations are often 30 days with a releasable product at the end of each sprint. The main advantage of Scrum is one of its key principles, and is the concept of "requirements volatility", it represent the idea that the requirements may drastically evolve during the developing process.

- **Extreme programming:** (3) The key elements are collaboration and quick and early software creation. It is founded on four values: communication, simplicity, feedback and courage. It usually conforms frequent releases in short periods of time. The main singularities of this method is the inclusion of the concept of pair programming and the prioritization of the tasks is strictly related to the actual need of it to be completed.

- **Kanban:** The last one studied is Kanban (4) methodology, it includes several differences among its competitors. The main non similarity is that Kanban does not specify the duration of the iterations and the schedule may contain only one large iteration which will cover the whole developing process. The

tasks are represented by cards and arranged in a board in several categories to provide a general snapshot of the status of the developing process. Each task has no predefined duration in the developing process, and can be completed into several stages moving from one category to another several times.

After analyzing the 3 main methodologies there were some conclusions. The first one is the versatility of the 3 methods because, although specifying some ideal parameters, they can actually be modified depending on the project requirements and the team involved. However, in a research project, the iterations duration estimation is not realistic and may incur into several changes in the schedule, due to the fact that a technology might not be usable for the final objective of the project.

The attributes of this project were more kanban related, as it could only be composed of a large iteration, however the ability of arranging and prioritizing the tasks will provide at every situation a general eyesight of the actual progress. It has no predefined start or end time of each task, which precisely represent the research task characteristics.

The task organization was made in "Trello" platform, which provides a clean and simple user interface. The categorization was splitted in 4 columns:

- To do: Where all the defined task are created and tagged. It represents the work backlog pending to be started.
- In progress: Cards in this category are under development. Depending on the stage, several cards could be in this category at the same time due to dependencies between tasks.
- Done: The tasks which initial development process has finished are moved to this category. Cards in this column may move to "In progress" if the later tasks or analysis require some kind of change or improvement.
- Completed: Tasks in this column are finished and tested, and instead of deleting them,we will maintain this category for organization purposes.

For further organization, all the tasks were divided by four categories:

- DJI: All the related tasks using DJI SDK code or analysis.
- Neural Network/Artificial vision: This category encompasses the neural network analysis, design and coding related to artificial vision.
- Documentation: Required changes and additions to the documentation. As all the other categories would also be included in the documentation, this tag was only assigned when it was required to clarify this assumption.
- Analysis/schedule: This category contains all the cards related to some thinking process that would imply a change in the schedule of the project or add some new items to be developed.

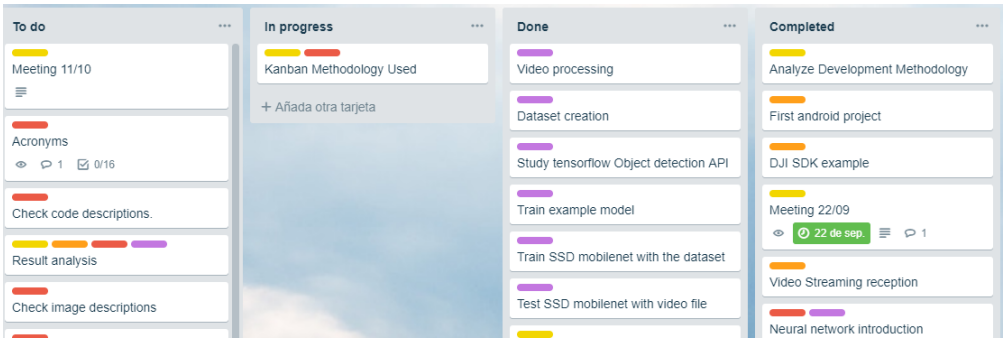


Figure 1.1: Task list in trello

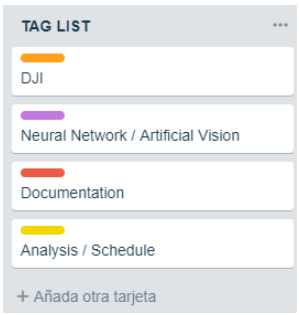


Figure 1.2: Categories defined in trello

As we could see in the figure 1.1 each card could contain one or more tags that represent its category because there might be an analysis task related to neural networks.

Using kanban methodology, the cards were created in Trello by following a schedule process. Notwithstanding several modifications were made during the evolution of the project.

1.2 Video processing

In this section of the project we will describe the method followed after receiving the video from the aircraft. We will process the video making use of Artificial Vision techniques with an Artificial Neural Network.

1.2.1 Object recognition

The task consists on being able to detect and recognize different kinds of aircraft in an image.

The field of object recognition has improved significantly in the last years. For example the face detection problem has been successfully addressed with "simple" algorithms like Viola Jones (5).

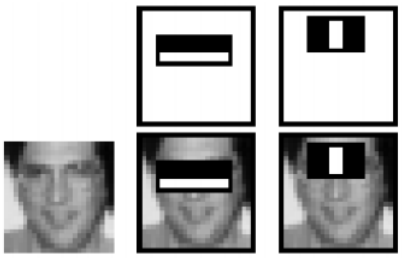


Figure 1.3: Viola Jones Algorithm principle.

An example of this is that every photography camera uses an algorithm that simplifies the process of focusing the image by detecting the faces.

We can also consider the objective of detecting people in an image almost achieved, an example of this is the Histograms of oriented gradients for human detection (HOG) (6). And the most important contribution this field has been the use of Convolutional Neural Network that will be explained in the section 2.2.

The main issue to confront when working with this method is the identification of reduced size objects like a drone. There are some articles trying to address this complication, and we can find some procedures to detect flying object (7), but in this scenario the only goal of the algorithm is to detect drone movements in the image. If the drone stops moving we could not be able to detect it anymore.

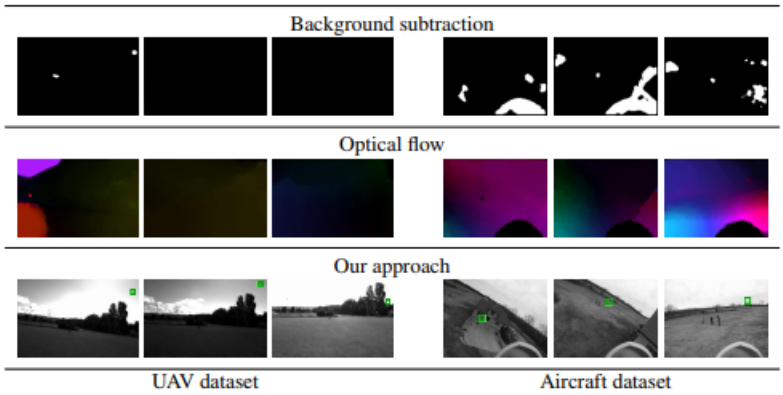


Figure 1.4: Flying Moving Objects paper proposed method results.

As we are discussing, the principal difficulty of detecting this kind of objects is that, in general, the drones physical dimensions are pretty reduced and the shapes may vary significantly from one model to another. It also depends on the manufacturer of the aircraft but the common size is frequently smaller than 50 centimeters. In this project we have been implementing the software in an aircraft which dimensions are the following:

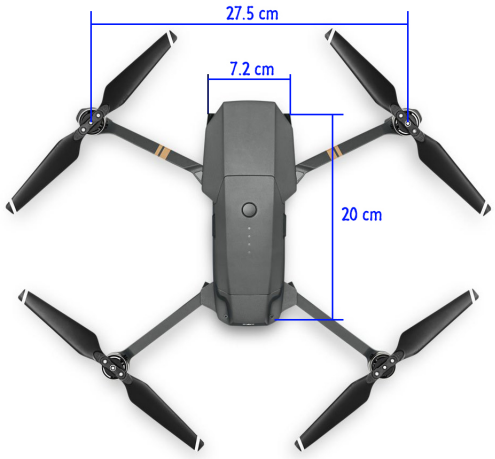


Figure 1.5: *DJI Mavic PRO* dimensions

With this physical characteristics, the identification task is really difficult, due to the fact that at a determined distance, the information that we can obtain from the image is reduced to a very small amount of pixels.

The procedure that we are going to implement is different to the technique of detecting flying objects (7) because we will not analyze the behavior of the

drones or estimate the movement. Instead of this, we will use a Convolutional Neural Network 2.2 in order to learn the physical characteristics of the aircraft, for example attributes like color, shape, borders, contrast with the background will be useful to achieve this objective.

1.2.2 Artificial vision

The artificial vision field has been increasing its popularity since the century beginning caused by the rapidly improvements in the processing capacity of the actual computer systems.

This performance increase made possible the implementation of a series of improvements in the used algorithms. An example of this application are the Convolutional Neural Network, which are an evolution of the Conventional Neural Network.

We will begin by introducing the concept of Artificial Neural Network and we will cover the major part of the most recent concepts in the next chapter.

1.2.3 Developing process analysis

After analyzing the problem faced, the development process, due to this fact, will consist on the creation of a dataset which will contain images of all the types of drones that we want to identify.

The amount of pictures needed will vary depending on some characteristics of the object to identify, like the complexity of it, the neural network model used and the different sides of it (i.e. the object viewed from the front is completely different to the same object viewed from the back). The cause of this last sentence is because if the object viewed along its faces is really different from one to another, it will be more difficult to the neural network to learn in order to identify this object. Due to these facts, the size of the dataset can not be precisely estimated.

Once we consider the dataset contains sufficient data, the next step in the development process is labeling the images, this step consists on generating a text file per each image on our dataset, in which we will specify if any of the object we desire to detect is present on the image. If it is the case, we will generate a new entry on that text file which will consists on the name of the object, the x and y coordinates of the object in the image and its width and height in pixels. We will generate as many entries as objects present in the image.

After the labeling process is completed, we will start to train the chosen Neural Network model. The training process of a Convolutional Neural Network designed to detect objects is the same as others. It is recommended (if not mandatory) to split the dataset (8) in "train" and "validation" folders, so that we can train the network with the images of the train split dataset and validate its results with the validation split dataset so we can monitor the training process and plot its evolution. The purpose of the validation splitting method is because testing the Neural Network behavior with the same images that we have used for training is counterproductive caused by the fact that it will learn to identify the objects in the

dataset considering the same position, rotation and light conditions as the pictures that compose it, and the neural network will not produce a generalization that we need to achieve to identify the desired object under any circumstances.

The splitting method that we are going to use is quite straightforward as we will only shuffle the image dataset and split it into 90% which will be used into the training process and the 10% left for validation purposes. However, there are more sophisticated methods like K-fold cross validation (9) and leave one out (10) which are more convenient ways to perform the dataset splitting.

We will introduce the concepts needed to successfully implement the described methodology in this section.

Chapter 2

Neural Networks

In this chapter we will study the concept of Neural Networks.

We will start by introducing the notion and history of Neural Networks, followed by the explanation of the Convolutional Neural Networks.

After that, we will study the most recent advancements in the Neural Network field in order to analyze the best model for our application, this section will include the statement of adapted models, MobileNets and the Neural Network Single Shot Multibox Detector (SSD).

Finally, we will precede the necessary steps to be taken in order to set up the Neural Network training environment.

2.1 Artificial Neural Networks

Neural network (11) are computational models which inception took place on the middle part of the 20th century. At a glance, this information might sound quite unusual because of the fact that a technology stated 60 years ago would become one of the most fundamental pillars of the modern computing with only a few improvements of the original model.

Neural networks are composed by a set of neurons (nodes), each neuron contains an activation function which depending on an input computes an output by applying its function. The interconnection of this set of neurons provides us the ability of accomplishing solutions which would be really difficult or even impossible to achieve using other mathematical models.

The architecture of a Neural Network is based on 3 main elements, the input layer, the hidden layer and the output layer. The input layer is where the data is introduced into the system, the hidden layer part is where the data gets computed and interconnected through the different hidden layers. Each neural network might contain several hidden layers which will be composed of a set of neurons. The final part of the architecture is the output layer where the activation of the different output neurons is achieved. The combination of the activation in the output layer after the data is introduced in the input layer is what gives us the result for the problem for that determined input.

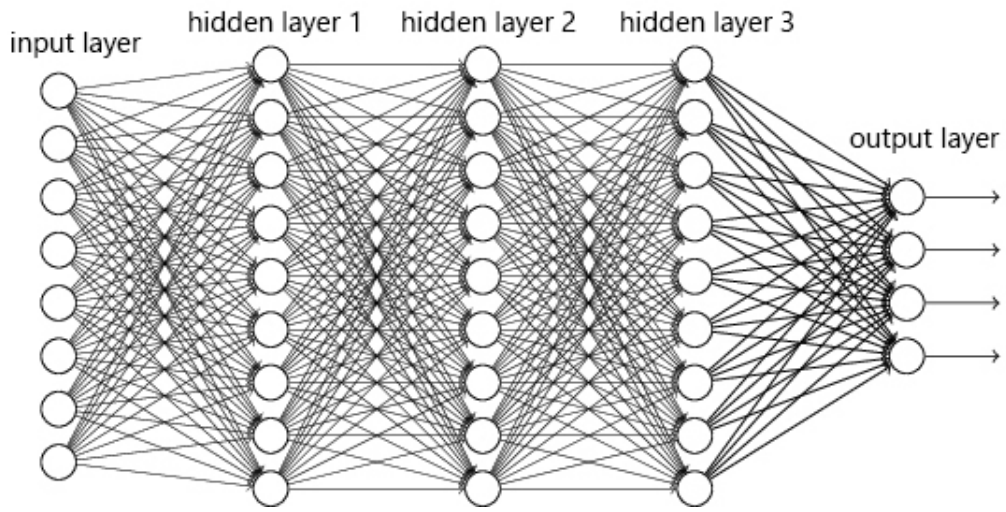


Figure 2.1: Neural Network with 8 input neurons, 3 hidden layers with 9 neurons each and 4 output neurons.

The quantity of neurons in the input layer is determined by the data we want to process with the Neural Network. The number of neurons in the hidden layer and the amount of layers can be estimated but usually the right combination is found after testing several parameters. The number of neurons in the output layer depend on the representation we need to achieve. For example, to solve a categorization problem we will have as many neurons in the output layer as categories we want to classify.

This structure has not evolved since it started until some years ago. Despite the fact of adding small improvements and variations to the original model, the principal drawback of Neural Network is that it requires a lot of resources to train and execute the model.

In the 80s, some fundamental changes were included and induced some improvements in the Neural Network architecture. The most important change is the concept of backpropagation.

2.1.1 Backpropagation

The concept of backpropagation (12) is commonly used in this days, it is a fundamental evolution which consists on introducing a set of data in the input on the neural network and propagating this data over all the layers by computing the output in each neuron with the objective of obtaining a result on the output layer.

This result in our network is then, compared with the expected output with a determined data input. With this comparison we generate an error value depending on the fact that the output is correct or if it is wrong, the difference between the output and the expected result.

This error is then propagated from the end to the beginning, starting on the output layer to the intermediate layers until we reach the input layer. With this behavior we compare the change that each neuron has contributed and the parameters of the neural network get adjusted according to the error that has been committed.

The backpropagation method is executed repeatedly until the desired loss metric is achieved thus it provides us the ability to adjust the parameters of the intermediate layers depending on the error rate that each one of the neuron have caused.

After this definition we need to analyze the differences of the traditional with regards to backpropagation. The main disadvantage of the traditional method was the size of the neural network, being the main reason the fact that the more neurons and layers a neural network had, the less likely it was that an error would produce a change, causing the error rate not to improve as efficiently and in some cases becoming stagnant. This behavior is known as weight decay (14). With the use of backpropagation, we manage solve this problem that caused the evolution of Neural Network to be lackadaisical.

2.2 Convolutional Neural Network

When working with Neural Networks it is important to mention one of the most important contributions to this area, the concept of Convolutional Neural Network (CNN) (13) since the characteristics of this kind of networks make them ideal for the artificial vision field.

These Neural Networks are a type of Deep Neural Network which are based in 3 fundamental parts. Convolution layer, pooling layer and classification layer. We will introduce these concepts and analyze what is an image in the following section.

2.2.1 ¿What is an image?

The important point we have to highlight is the representation of the image, which is composed of set of pixels with a determined value and each pixel is located in a fixed position of the image. We can represent this image as an input of a Neural Network by following the next schema:

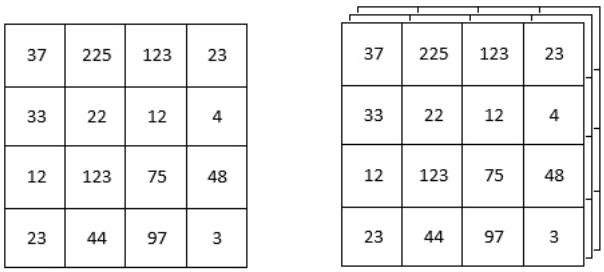


Figure 2.2: 4x4 image with 1 channel on the left and 3 channels on the right.

In this figure there are two 4x4 image, the one on the left side corresponds to a gray scale image, with only one value per pixel that represent the intensity between 0 and 255, being 0 white and 255 black. By the other hand, the image on the right could represent a 4x4 RGB image composed by 3 channels, where every channel corresponds to one of the basic colors (Red, Green and Blue). The combination of the 3 layers gives a color image.
Once this concept is clear, we can introduce the concept of convolution.

2.2.2 Convolution

A convolution (15) is the operation made in this type of Neural Network. It consists on a transformation that preserves the spatial relationship between the pixels whilst it learns of small fragments on the original image.

With this we introduce the principal difference between the Conventional Neural Networks introduced on the section 2.1 and it is the principal motivation of the Convolutional method.

In a Conventional Neural Network, the neurons on the input layer are connected with the layers on the hidden layer which then connect to the neurons on the output layer. The main problem with this behaviour is that it is not scalable to use with relatively big images due to the big quantity of irrelevant connections that would be generated.

Considering a 32x32x3 color image (32 pixels width and 32 pixels height), we will need to introduce 3.072 weights amongst all the layers. If we now consider a 200x200x3 image we will need to include now 120.000 weights, which is a huge quantity of unneeded parameters that only increase the complexity of the network and could lead to overfitting and falling on the problem of WeightDecay (14).

However, in the convolutional layers we introduce a significant change, instead of connecting all the neurons of a layer with the previous one, we perform a convolutional operation which follows the next schema:

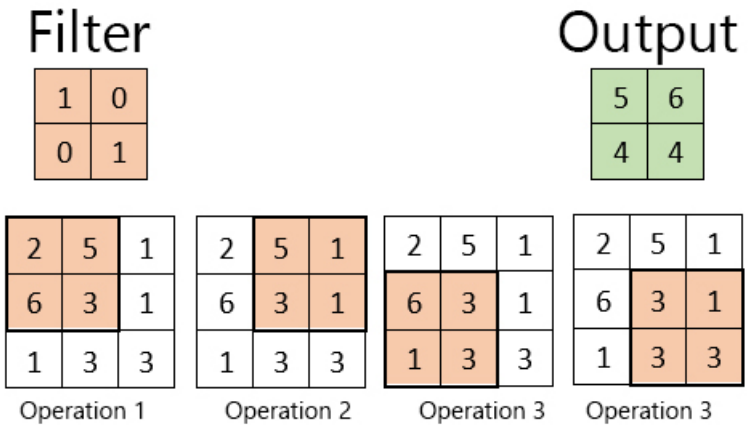


Figure 2.3: Simple convolution in a Neural Network.

As we can see this operation is like sliding a window along our original image. This window is a matrix named "filter" and has two size properties, width and height. In the example we use a 2x2 filter, and we perform 4 operations over the original matrix. The result of this operation is a new 2x2 layer.
In the convolution we have to specify the parameters stride and padding:

- **Stride:** By specifying the Stride (16) parameters we are configuring the

displacement that we will perform in the midst of each iteration of the convolution. Starting on the coordinate (1,1) and considering a stride of 2, we will skip 1 pixel in between each operation, the next iteration will be at the pixel (3,1), the following one at (5,1) and so on. If we have a stride of 1, we will apply this convolution in each pixel of the image, we will not skip any pixel.

- **Padding:** The Padding (17) parameter is a property added in order to not lose value in the margins of the image. This is due to the fact that when we are applying a convolution, if we have a 3x3 filter and a 30x30 image, we will start the convolution at the pixel (1,1), then at the pixel (1,2) and so on. We will realize that the value on the coordinate (1,1) has only affected to the convolution performed in the coordinate (1,1) whilst the value of the coordinate (1,2) has affected to the convolution on the coordinates (1,1) and (1,2), due to the fact that we are applying a 3x3 filter. We can observe this behaviour on the following figure:

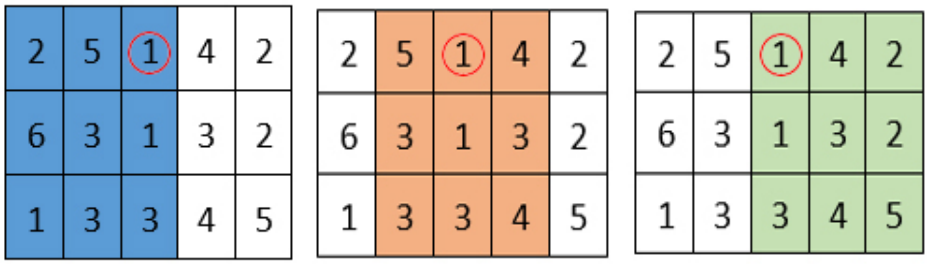


Figure 2.4: Convolution process example.

As we can see the coordinate (1,3) affects to the result of 3 convolutions, the coordinate (1,2) only in 2 and the coordinate (1,1) only in 1 convolution.

This is where the padding parameter offers some advantages, if we add a padding size of 1 which type is zero-padding (18) it would mean that we would add new components (pixels) at the ends of the original matrix with value of 0. We can see the effect of the Zero-Padding on the next picture:

0	0	0	0	0	0	0
0	2	5	1	4	2	0
0	6	3	1	3	2	0
0	1	3	3	4	5	0
0	0	0	0	0	0	0

Figure 2.5: Size 1 zero-padding example.

By using this method and combining different parameters, we will achieve the goal that the values on the extremes of the image are preserved. In this example we used the "zero-padding" method, but there are more examples depending on the context and the application needed.

- Convolution example:

In order to clarify these concepts, we can explain it by the following example:

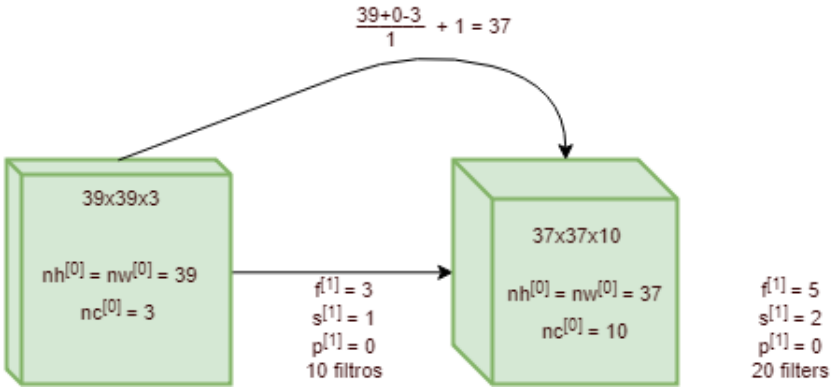


Figure 2.6: Convolution process example.

The input of the Neural Network is a 39x39x3 image, so its attributes are the following:

- nh = image height = 39
- nw = image width = 39
- nc = number of channels = 3

And we apply a convolution with the following characteristics:

- f = 3 Filter size.

- s = 1 Stride value.
- p = 0 Padding size.
- 10 filters Quantity of filters to use.

When applying this convolution, the attributes of the output after performing the convolutional method, will be determined by the following formula:

- Width and height of the convolution:

$$d = \frac{(n + 2p - f)}{s} + 1$$

  With this example, the output will be as follows:

$$d = \frac{(39 + 2 * 0 - 3)}{1} + 1 = 37$$

- Number of channels will be equal to the quantity of filters used (in this example, 10).

2.2.3 Classification layer

The final step to define a Convolutional Neural Network is the concept of Classification Layer. As we are going to make use of this kind of Networks to classify images, in the output of the neural network we need to obtain a value that will serve us to know in which class we can classify each of the images. To fulfill this purpose, we will add a new layer that will contain as many neurons as classes we want to classify. (e.g: if we need to classify objects in 10 classes we will have an output layer of 10 neurons).
In this kind of problems, we will usually be adding a layer that is called softmax (19). This layer is normally added at the end of all the convolutions, we can follow the above example until we get a convolution result of 6x6x80. After this layer, we will follow a method of linearization, we will unite all the parameters of the network linearly in a vector, so we will obtain a layer with 6x6x80 = 2880 parameters. This vector is then connected to a softmax layer, which will contain as many neurons as classes we want to classify. The schema of the described operation is the following:

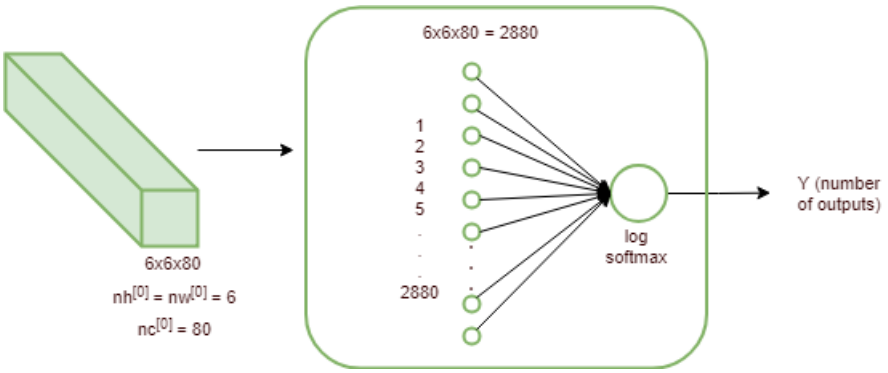


Figure 2.7: Softmax classification example.

2.2.4 Convolutional Neural Network example

One important example of the purpose of a Convolutional Neural Network is the network YOLO (20), which is a neural network which purpose is detecting and classifying objects whilst executing in real time on almost any reasonably new computer.

It is based on a human behavior, in which at a glance we can already discern the object that are located in any image, for this reason the network is named "You Only Look Once".

The operation principle is different from many detection models, because most of them process only a portion of the image and it is analyzed if an object exists in that portion or not. This is like a window that is moving around the image n times, executing the Neural Network at every step depending on the window size. YOLO is different because it processes the whole image, so it is only executed once every image. This caused a really important turning point in the object recognition models, since we are able to perform the detection of the objects in an image with a single execution of the Neural Network.

Example of the YOLO Neural Network method.

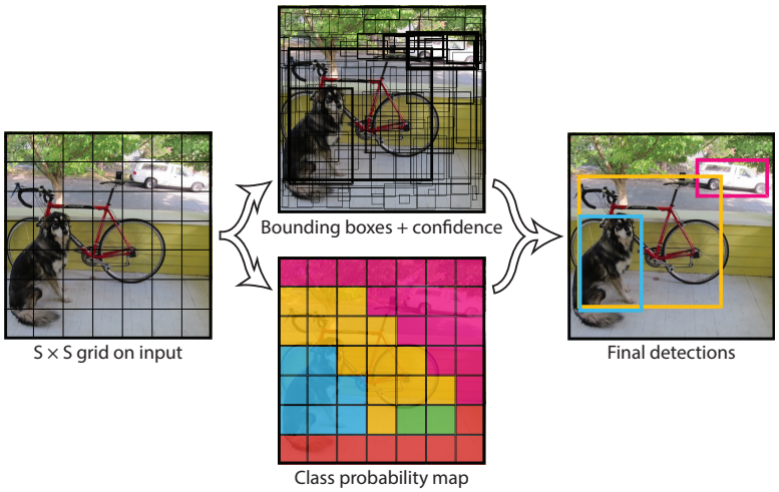


Figure 2.8: Process schema of the Neural Network YOLO.

We can observe that the image is divided in cells of size SxS, in each of the generated cells we made a prediction of some regions in which an object could be located. We introduce a "C" parameter, which is the number of classes that the Neural Network model is running on, so in each of the generated regions the probability that an object of each class is present in its boundaries is calculated. We also add a parameter "B", which determines the amount of regions that can exist in each cell.

For example, if we execute the YOLO Network with the following parameters:

- S = Regions size = 7
- C = Number of classes = 20
- B = Amount of regions in each cell

We will divide the image in a 7x7 grid, in each cell of the grid we will detect at most B (2) regions, and the number of classes (C) is 20. So we will calculate the probability of every class to be present on each of the regions that we obtain.

For further references, some revisions of the Neural Network have been published, the YOLOv2 (21) and YOLOv3 (22), which is the most recent version published on May 2018, which includes numerous improvements with respect to the initial Neural Network version.

2.3 Neural Network on mobile devices

Once the structure of a Convolutional Neural Network is understood, we must find a way to carry out our purpose of using a mobile device as the only computational method to process the data with the Neural Network.

Due to this limitation, several methods emerged, like the adapted method "tiny-yolo" (23) which is a reduced version of the model studied in the previous section. As a result of this, the model is executed faster but the error rate is higher than the full model.

We will discuss now the types of adapted models that exists nowadays.

2.3.1 Adapted models

There are some solutions that can be employed to process data on mobile devices, some of those techniques are the following:

- **Neural Network hashing:** (24) Consists on adding a layer that has the characteristics of a hash table. This lets us improve the efficiency of the neural network, especially when working with images, considering that all the attributes that are common to a type of object will be stored together following the structure of the hash table.

- **Neural Network Compression:** (25) The peculiarity of this method is that it make use of a previously trained model which is then subjected to an analysis of its learned characteristics. This consists in checking the weights obtained from the training and analyzing redundancies and repeated characteristics that could be potentially reduced.

- **Neural Network Pruning:** (26) Using the Neural Network Pruning method we can analyze the interconnections between layers of neural networks, to know if the link is completely necessary or can be dispensed if it does not provide enough information. This acquires a huge importance, because the large number of links that exists in the neural network can cause that unuseful operations are performed without adding a relevant change to the output. By eliminating this links we also reduce the amount of weights causing a reduction on the size of the model.

  In the following illustration the obtained results are shown, achieving a reduction performance up to 19X in some scenarios.

	LeNet-300-100			
	fc1	fc2	fc3	total
Weights	235K	30K	1K	431K
Pruning	94.7%	83.8%	11.9%	94.7%
	19×	6×	1×	19×

	LeNet-5				
	conv1	conv2	fc3	fc4	total
Weights	0.5K	25K	400K	1K	431K
Pruning	31.1%	82.2%	96.6%	41.7%	93.3%
	1×	6×	29×	2×	15×

	AlexNet								
	conv1	conv2	conv3	conv4	conv5	fc6	fc7	fc8	tot
Weights	35K	307K	885K	663K	442K	38M	17M	4M	61M
Pruning	3.8%	34.9%	24.3%	36.7%	33.5%	96.8%	91.5%	78.5%	91.5%
	1×	2×	1×	2×	2×	31×	12×	5×	12×

Figure 2.9: Neural Network prunning

- **Neural Network distillation:** (27) This method consists in using a big and complex Neural Network to train a less complex structure which can be any standard mathematical model. An example of this is the train of a soft decision tree by using a neural network.

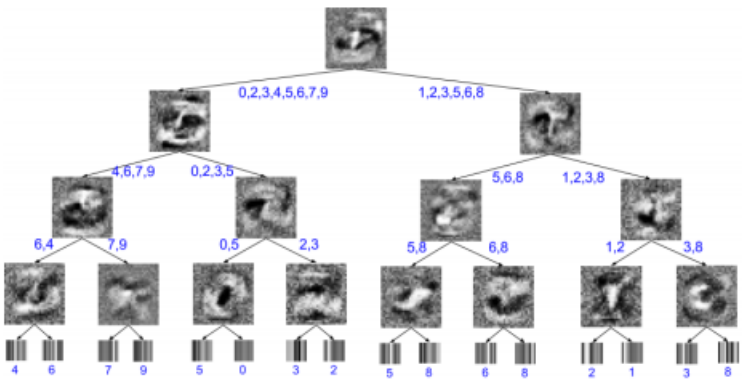


Figure 2.10: Soft-decision tree example from a Neural Network trained for MNIST.

2.3.2 MobileNet

Apart from these reduced and adapted models, we are going to introduce the concept of MobileNets (29) which are a set of optimized models designed to improve the efficiency on mobile platforms.

MobileNet is a really important contribution to the field of Neural Network, because it arises of the need to optimize the performance of Neural Network, prioritizing the efficiency of the model over other characteristics.

In order to achieve that efficiency, the model breaks down again with the already defined schemes and introduces some significant changes in the convolutional layers.

The MobileNet structure proposes to change the conventional convolutional layer by a "depthwise" convolution and a "pointwise" convolution, the characteristics are as follows:

- **Depthwise convolution.** We perform only one filtering process in each channel of the input layer.
- **Pointwise convolution.** This convolution is a 1x1 convolution, whose impact on the network will be studied before.

As we saw in the CNN 2.2 section, the convolutional layer filters and combines the inputs into a new set of outputs. The model "depthwise separable convolution" separates in two layers the convolution, one to filter and the other to combine the results; this let the network to drastically reduce the computational requirements and the weight of the model. The reason of that will be explained on the following section.

Computational cost comparison

In order to analyze the computational cost, we must start by obtaining the traditional convolution cost:

$$D_K * D_J * M * N * D_F * D_F$$

The computational load depends on multiplying the number of input channels (M), the number of output channels (N), the size of the kernel (Dk x Dk) and the size of the filter (DFxDF).

In the depthwise separable convolutions model, we break this relationship between the number of input channels and the kernel size, which is achieved by using the two layers previously described.

The depthwise convolution is used to only apply a filter on each of input channel. After this we apply the layer pointwise convolution, which consists in a 1x1 convolution to create a lineal combination of the output of the depthwise layer.

It is important to mention that after the two new convolutions, we usually include a Batch normalization (30) layer and a ReLU (31) layer.

The depthwise convolution layer with a filter for each input channel has the following computational cost:

$$D_K * D_J * M * D_F * D_F$$

With this convolution applied, we have only filtered the input channels, however we have not combined them to create new characteristics on the network.

To solve this problem, we add a new layer in between the depthwise convolution and the pointwise convolution, this layer's objective is to linearly combine the output of both layers. This linear combination of the two layers is then, called "depthwise separable convolution". The cost of this whole combination is:

$$D_K * D_J * M * D_F * D_F + M * N * D_F * DF$$

Which corresponds to the sum of the computational cost of both depthwise convolution and pointwise convolution layers. When we separate the convolution process in 2 steps of filtering and combination we obtain a time reduction which can be calculated by the formula:

$$Time\ reduction = \frac{Deptwhise\ separable\ convolution\ cost}{Conventional\ Convolution\ cost}$$

By substituting the cost in the calculation we obtain the following

$$\frac{D_K * D_J * M * D_F * D_F + M * N * D_F * DF}{D_K * D_J * M * N * D_F * D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$

MobileNet architecture introduces this convolutions of size 3x3, so it requires 8-9 less time than a standard convolution to calculate each iteration.

To summarize, the depthwise separable convolution schema in comparison with a traditional convolution is the following:

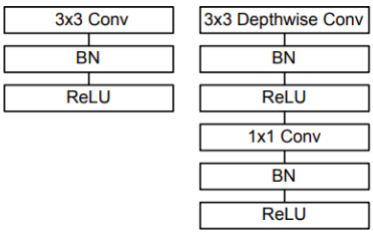


Figure 2.11: Depthwise convolution compared to a traditional convolution.

Result analysis

We can observe that in the obtained results we can appreciate a substantial diminution on the parameters number of the model, while maintaining an accuracy value similar to others.

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Figure 2.12: Model comparison with the current state of art.

Another important metric that we can analyze is "Million Mult-Adds", which indicates how many arithmetic operations are performed by the Neural Network. The comparison is the following:

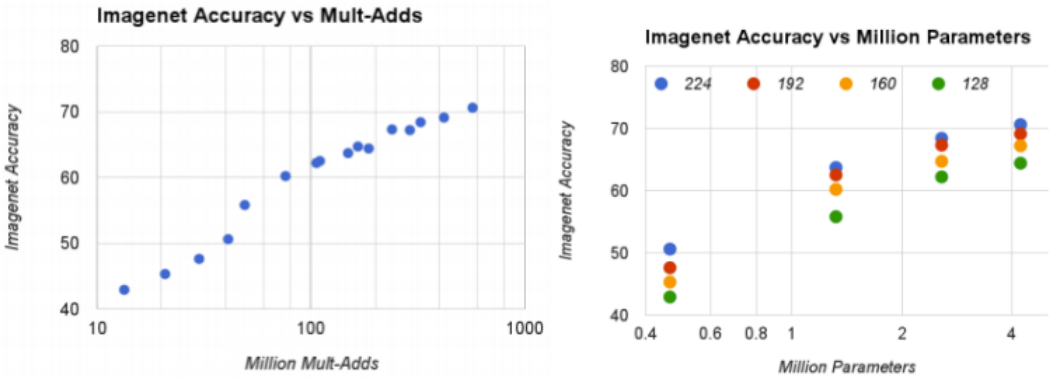


Figure 2.13: Million Mult-Adds metric comparison.

Firstly, with the relationship of Mult-Adds and accuracy we can obtain that the best point considering both factors is around 100 million Mult-Adds, and we can also observe a relationship between Accuracy and millions of parameters, where the increase in the accuraccy is notorious in between 0.4 and 1.5.

2.3.3 Single Shot Multibox Detector

When investigating more deeply the concept of MobileNet we come across to the Neural Network that is going to be used in the development of the application of this project.

The name of this model is SSD (32), which acronym represents "Single Shot Multibox Detector" and can be translated as a Neural Network that only analyzes the image once, so this implies that both detection and classification are performed at the same time.

The model is based on the Neural Network VGG-16 (33; 34) which is a type of model that is classified as very deep neural network.

It was firstly developed to analyze the impact of deepness on the Neural Network, and how it was related with the accuracy of the model. The strategy of

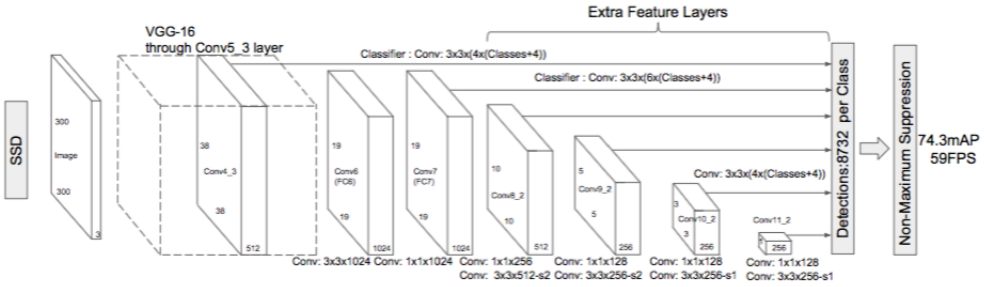


Figure 2.14: SSD Model schema.

this Network is using reduced size convolutions (3x3) to drastically increase the deepness of the network. The VGG16 is composed of 16 layers in which the corresponding convolutions are performed.

The principal difference between this and the VGG-16 model is that it does not include the fully connected layers, which we can see highlihted in blue.

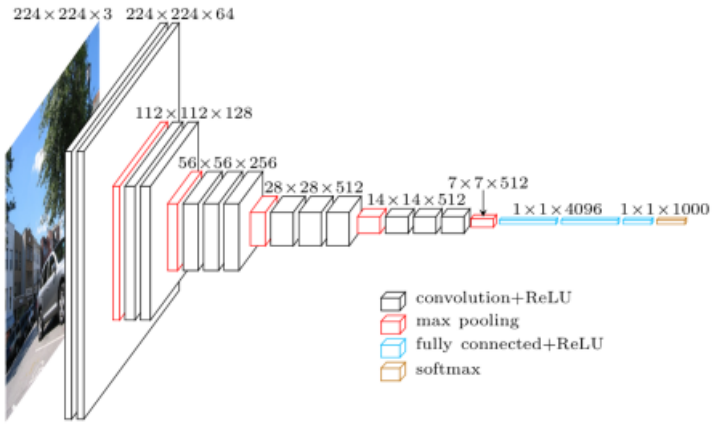


Figure 2.15: VGG-16 model schema.

Multibox

With this model the concept of Multibox (35) is introduced, which is inspired by an algorithm with the same name.

It is a method that produces a fast analysis of the candidate areas where an object might be present. This Multibox algorithm adds two new parameters to the model:

- Confidence Loss: It is a metric that measures the likeliness of an object to fall inside of a calculated region.

- Location loss: The distance between the calculated region and the real region specified in the train data.

$$multibox\_loss = confidence\_loss + alpha * location\_loss$$

By using this formula and introducing an alpha variable, we can modify the weight of the location loss to the calculation of the multibox loss, which will be used in the training process to minimize this loss.

SSD model

After introducing the concept of Multibox, we are going to analyze the operation principle of the model.

The Neural Network model receives an input image, and making use of the results obtained in the Multibox, a series of regions of different sizes are evaluated. For example, 4x4 or 8x8 regions which is called "feature map".

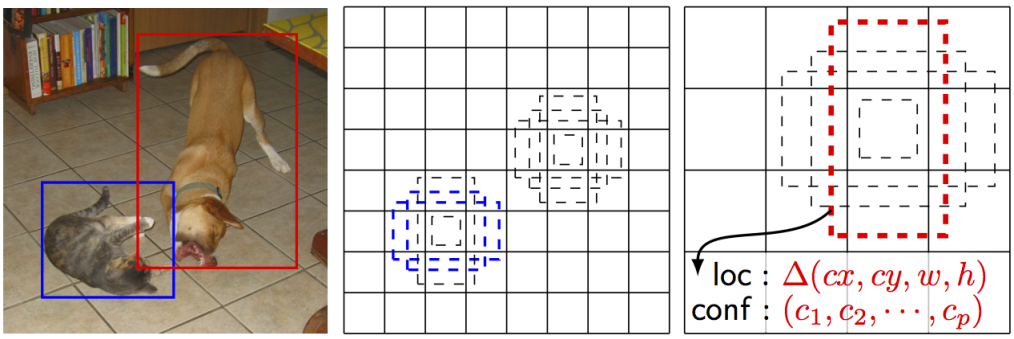


Figure 2.16: SSD Neural Network Feature map.

We can clearly see the advantages of this method in the picture above. With a smaller feature maps (e.g 4x4), which have fewer regions, the objects that are larger will be highlighted with greater precision, while the smaller the divisions of the regions, the better the smaller objects will be detected (e.g 8x8 feature map).

The network analyzes how likely it is to a region to pertain to any of the classes that we want to detect, and this prediction is made in all the generated regions.

By using this procedure, we can extract the following behaviour: In order to detect 3 classes, we will obtain in each of the regions, the probability of the region to pertain to a determined class (1, 2 or 3). In this way, the regions that have the highest probability are combined with the regions generated by the MultiBox. Therefore, what we obtain is a result that combines the regions generated by the Multibox in addition to the probability of an object of any of the classes to fall in each of the regions.

With this combination we obtain the confidence loss that we stated in the previous formula:

$$multibox\_loss = confidence\_loss + alpha * location\_loss$$

Consequently, what we obtain by applyting this method is separate the behaviour of detection regions (delegating this activity to the MultiBox) and analyze the probability of an object to be inside of any of those regions.

2.4 Neural Network training environment

Once the concept of Neural Network is defined, one of the most important characteristics of them is the ability to train a model to perform a slightly different task than the purpose it was created.

We are going to exemplify this process by using the Neural Network model Single Shot MultiBox Detector (SSD), which was subject of study in the section 2.3.3.

With the purpose of explaining the training process, we need to specify the work environment we are going to be using.

2.4.1 Work environment

In order to perform the training process, we are going to be using the operating system Windows 10, although this process can be done in several environments, this was chosen because of the compatibility between some of the programs and frameworks that we are going to describe in the next sections.

2.4.2 CUDA

The first step to setup the environment, is the CUDA (36) library, which acronym represents "Compute Unified Device Architecture". It consists on a platform of parallel computing which uses the power of a graphics card to perform a series of defined tasks.

By using this architecture, we obtain a boost in the performance of the models, due to the fact that we make use of all the optimizations of the library by using the graphics card of our computer which implies a lot of advantages over the use of a CPU. We will introduce the reason of this advantages.

Graphics Cards have experienced a huge improvement in the computing capacity. This improvement can be measured in FLOPS (37), which is the acronym that represents "Floating Point Operation per Second" and can be used as a metric in contrast of IPS (Instructions per Second) which is often measured with MIPS (Million Instructions Per Second). The importance of FLOPS in this field is its application to scientific computing, because the amount of floating-point calculations is significantly larger than in other applications.

The improvement experienced was notorious since 2009 due to the fact that Graphics cards' performance was around 650 to 1000 gigaflops in that year and it continued rising to some values of 3000 gigaflops in the year 2012.

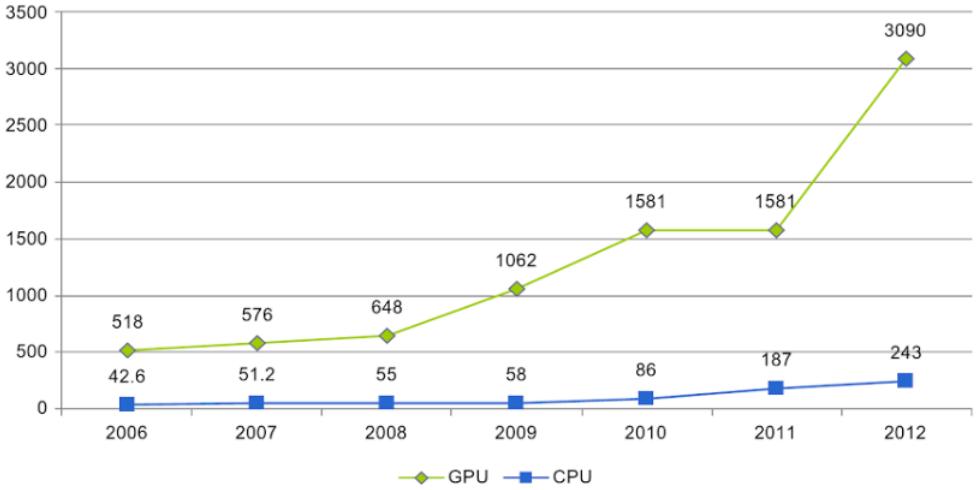


Figure 2.17: CPU and GPU GigaFlops comparion. Obtained from reference (36)

This enhancement in the computing capabilities of Graphics Cards was caused by the inclusion of "massively parallel hardware". For instance, the GPU Nvidia G80 included 128 CUDA cores, when compared with the "Fermi" architecture with 512 CUDA cores, a jump in 300 gigaflops was achieved.

The use of CUDA library, provides us the needed tools to take all the profit from Graphics Cards' parallel computing capabilities. The used version of the library is 9.0, due to the compatibility with the Tensorflow version to be used.

To summarize, the use of this architecture is practically essential for the training process of Neural Networks, since CPU training requires much longer time than required when using the CUDA library training on a GPU.

In the next figure we can see some data of training comparison on a CNN 2.2 on Cifar-10 dataset (38), the training process was made in 4 different setups:

- CPU 2xAMD Opteron 6168 1.9Ghz (2x12 cores).
- CPU i7-7500U, 2.7Ghz
- GPU Nvidia 940MX 2GB
- GPU Nvidia GTX1070 8GB

Device	Speed of training, examples/sec
2 x AMD Opteron 6168	440
i7-7500U	415
GeForce 940MX	1190
GeForce 1070	6500

Figure 2.18: Training data metrics obtained from this article

The results are pretty straightforward to compare, with a reasonably cheap GPU as the Nvidia 940MX the results obtained are 2.5 better than the performance obtained from a normal CPU.

In case we use a medium budget GPU like the Nvidia 1070, the results obtained cannot be matched by a processor.

2.4.3 Python

As a consequence of the integration of Python programming language with the different Deep Learning frameworks, like Keras, Tensorflow, Caffe and others. Followed by the ease of use of arrays caused by the inclusion of Numpy (39) library, and the direct and simple way of working with images thanks to the PyPI (40) (image-utils) library, is the reason of Python to be one of the most important languages in the field of Neural Network projects.

Specifically, the training process will be carried out in the Python 3.6.0 version, which can be downloaded from the official website at https://www.python.org/

Once downloaded and installed, the package manager PIP is included in our system, so we will use it to download the following dependencies:

- **Numpy:** Numpy (39) library is fundamental if we are going to work with multi-dimensional arrays, its a mathematical library which adds several high level functionalities to Python, which are really useful when working with Neural Network models.
- **Scipy:** A library focused on scientific computation which includes some lineal algebra functions, image processing, interpolations, and more. Which is designed to work fast and efficiently with numpy arrays, so the integration between these two libraries is pretty convenient.
- **Scikit-image:** Scikit image (41) provides some techniques to work with images. For example, image segmentation, geometric transformations, filtering techniques, etc. Due to its ease of use it becomes an essential tool for the development of this type of applications.
- **Jupyter:** Jupyter notebook is based on a web application, which can be used to execute Python code and share it easily and intuitively. We will be

using this library in the first stages of the Neural Network training, in order to execute some needed examples.

- **Tensorboard:** We will be adding to the Tensorboard library, which provides us a real time dashboard which will be monitoring the Neural Network training process plotting some important charts with the related data. Its simplicity to install and work with the obtained data makes it ideal to work when training Neural Network models.

After installing these Python dependencies, the next step is the introduction of the library we are going to be using in the Neural Network training process.

2.4.4 Tensorflow

Tensorflow framework is developed by Google Brain Team and combines several computational algebra techniques with model Neural Network calculus optimization, thus facilitating the calculation of different functions needed by the models that under other circumstances will require a lot of time to carry out.

It is the base of other frameworks that provide a high level abstraction of its methods and add some extra functionalities which can be really useful depending on the application. Its popularity is granted due to the simplicity of working with several previously designed Neural Network models, thus providing different training techniques depending on our requirements in addition to the huge amount of functions and method we can access to modify or create our own model.

Its key features are the following:

- Definition, optimization and efficient calculus of mathematical operations with multi-dimensional arrays (introducing the "tensor" concept).
- Support for the majority of the techniques used in Neural Network training, for example regression functions, layer models, optimizers, dropouts, etc.
- The calculus operations are performed in an autonomous way to the user, so Tensorflow will analyze what is calculated on the CPU and what will be computed using the GPU.

All these functionalities are the key features that makes Tensorflow become an exceptional library in the Neural Network developing process, centering on the important pieces of the creation, modification and training of computing models.

Another key factor of the framework, is the free and easy access to a huge amount of tested models which are maintained by both the Tensorflow team and the community. The models are available in its Github repository (44).

We will be working with the Tensorflow-GPU version. After installing Python, using the package manager pip we will introduce the following command "pip install tensorflow-gpu".

2.4.5 Tensorflow detection API

In order to develop the application, we are going to use the Tensorflow detection API.

The reason of this decision is because of the type of needed application. It is the best option due to the fact that it works over Tensorflow, and provides several examples and pre built models that are convenient for object detection projects.

We will start by executing the example code provided by Tensorflow, we will clone the repository https://github.com/tensorflow/models

After this step is completed, we will navigate to the path: "models/research/object_detection" and execute the following command: "jupyter notebook"

As we mentioned previously, jupyter notebook helps us to easily create a workspace to test and develop in a intuitively way. The notebook provided by Tensorflow is well commented and explained, so it is easy to follow.

Once executed we obtain the following result:

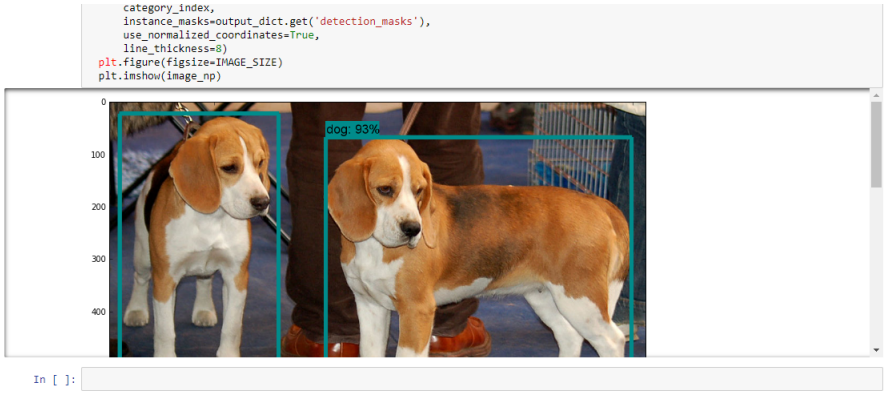


Figure 2.19: Jupyter notebook execution.

After this small test, that serves us to check if all the dependencies are working properly, we can proceed to the process of training the desired Neural Network by using the Tensorflow detection API and the SSD 2.3.3 model.

Chapter 3

Application development

We will follow in this chapter the development of the application that will run on an android mobile device.

We will start by defining the architecture of the system by introducing some UML modeling diagrams.

Followed by this section, we will define the communication schematic followed to receive the video streaming from the Aircraft.

Moreover, we will also carry out the Neural Network training process besides to the introduction of the method of running Neural Networks on Android.

Finally, we will add a functionality to the application of commanding the aircraft autonomously by using the detections obtained by the Neural Network

3.1 System architecture definition

In the first section of the implementation chapter we will cover the introduction to the application architecture.

We will start by introducing the concepts of the Unified Modeling Language (42) owing to explain the used methodology to define the system.

To conclude with this section, we will explain both class diagram and use case diagram, because due to the reduced complexity of the architecture of the developed program, we will only use these two types to define the system.

3.1.1 Unified Modeling Language

The Unified Modeling Language (UML) (42) provides a standard software design principle whose importance and benefits have been widely demonstrated.

With the use of this modeling language we achieve to define the architectural elements that form our application, focussing on the high-level interactions between the defined elements by switching from code-based development to architecture-based development.

Class diagram

Class diagrams (43) have become the most common diagram in the object oriented system modeling. It is a diagram that contains all the classes, interfaces and collaborations along with the relationship between them.

We use them to provide a static view of the system, defining the vocabulary and the collaboration.

The designed class diagram, contains some classes that will be defined in the sections. The ConnectionActivity, DJIVideoStreamDecoder and MainActivity will be defined in the section 3.2, specifically sections 3.2.3, 3.2.3 and 3.2.3 respectively.

The class DJIDetectorActivity will be defined on the section 3.4.

Finally, the class CommandAircraft definition can be seen on the section 3.5.4.

The resulting class diagram is the following:

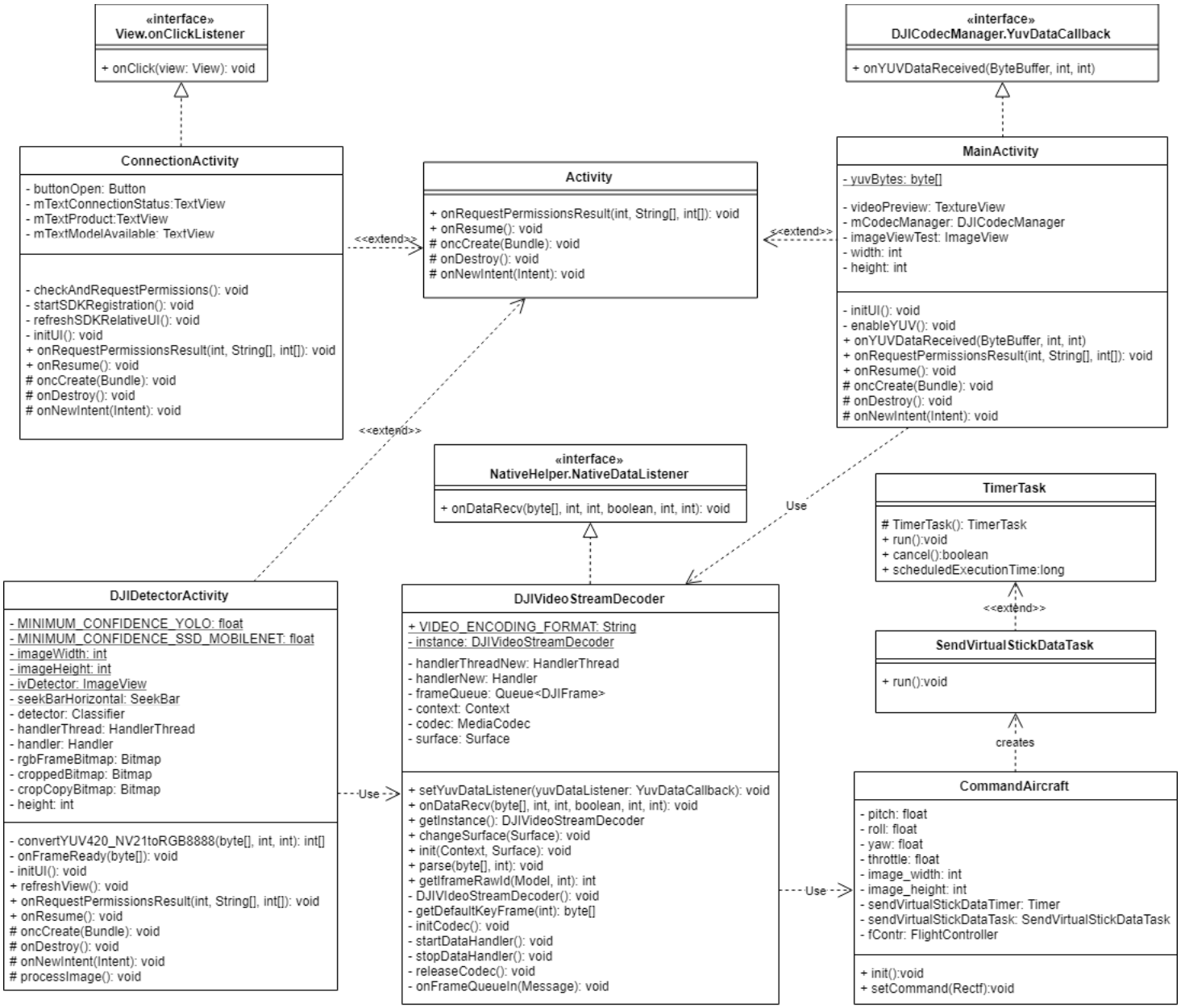


Figure 3.1: Class diagram

Use Case diagram

We use a use case (43) to describe a set of predefined actions which can be started by an actor that can be performed by a system and can result to an output to a particular actor. This diagram bases its importance in providing a description of the behavior structure of the system.

We will define our system with 2 diagrams both representing a use case. In the description we find the following information:

The first one describes the application launch, the steps needed to authenticate the application and obtain the aircraft information to start the application.

Use case: Entering the application	
Actor: User of the application	
Normal execution	Alternative execution
1) The user starts the application. 2) The application is authenticated in the DJI server. 3) The user connects the aircraft via USB. 4) The system obtains information about the aircraft. 5) The user opens the main view	4.1) Show error message if not able to obtain information.

Figure 3.2: Use case description 1

The diagram is the following:

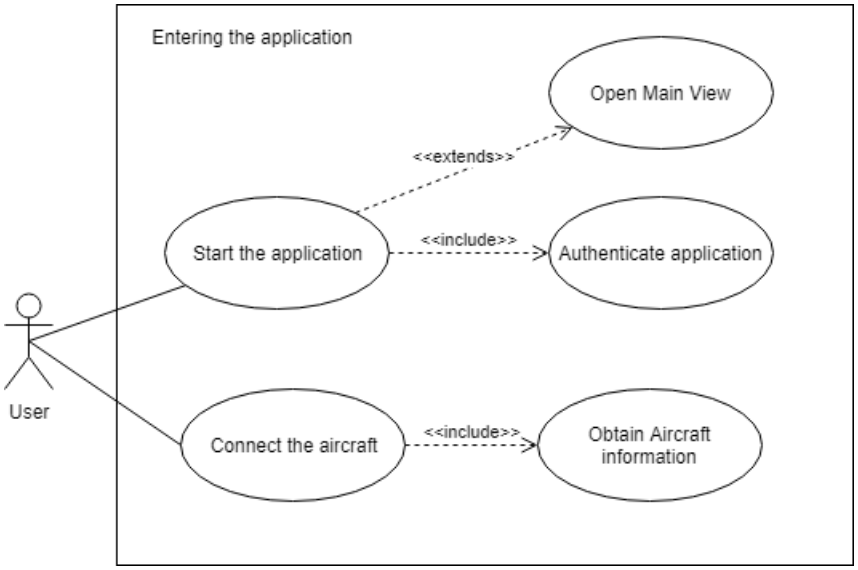


Figure 3.3: Use case diagram 1

The second diagram is used to describe the execution of the detection part which starts by the user starting the view, followed by the execution of the video streaming decoding and the creation of the detection thread. The user can start the commanding mode where the application will start commanding the aircraft depending on the position of the detected object.

Use case: Executing the detection view	
Actor: User of the application	
Normal execution	Alternative execution
1) The user opens the detection view. 2) The application starts the video streaming decoding. 3) The application starts the detection thread. 4) The user starts the commanding mode. 5) The application commands the UAV depending if there is an object detected and considering the object detected position.	

Figure 3.4: Use case description 2

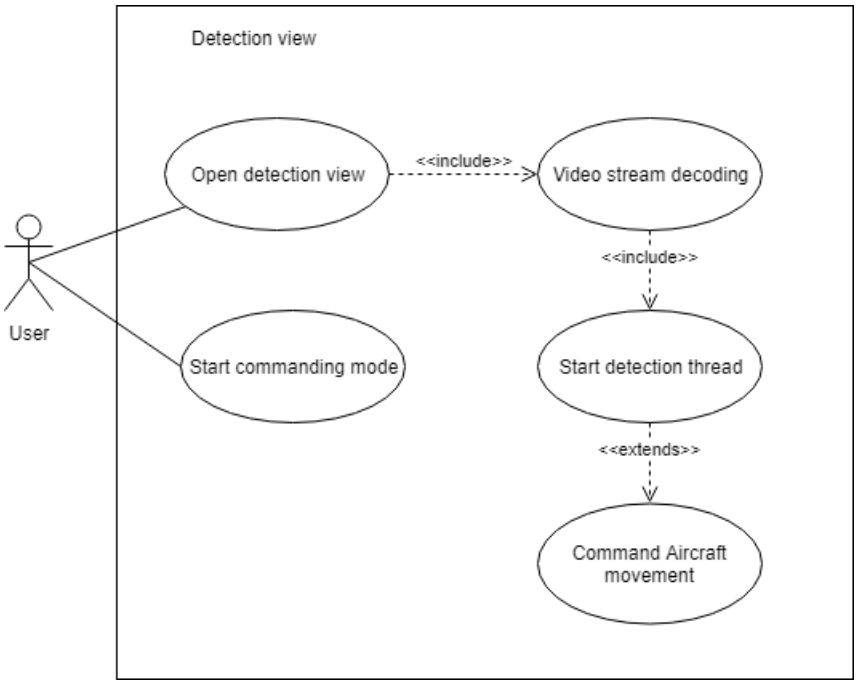


Figure 3.5: Use case diagram 2

3.2 Video streaming reception

This section defines the steps needed to be taken to receive the streaming from the DJI Mavic PRO *drone* in some *smartphone* in order to process this streaming with the artificial vision algorithm to finally show the results on the screen.

First we should analyze the components that are needed to achieve this purpose and understand the communication interface between each one of the components to be able to receive the video feed from the *drone*.

3.2.1 Communication schematic

The communication schematic that we are going to use is the following:

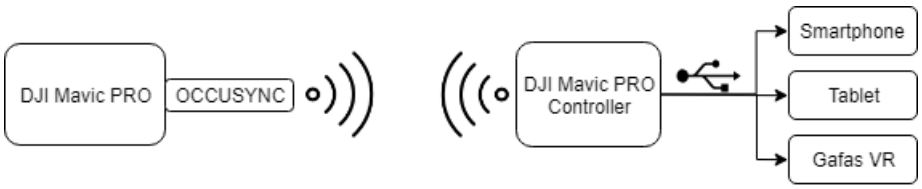


Figure 3.6: Video and data connection schema with the drone DJI Mavic PRO

In the picture we can see some components that are described below:

- **DJI Mavic PRO:** The aircraft that we are going to use to develop the project. Even though we could use any DJI brand *drone* that had USB connection, due to the fact that the video streaming from the different platforms, although the transmission technology may differ, the video reception is exactly the same because of the use of the DJI SDK.

Figure 3.7: DJI Mavic PRO front view

- **OCCUSYNC:** Transmission technology used in the DJI MAVIC PRO. It offers data link distances up to 7km.

- **DJI Mavic PRO Controller:** The remote controller (RC) used to command the *drone* and receive the video feed in real time from the camera.

Figure 3.8: DJI Mavic PRO Controller

- **USB connection:** This step is optional but we will in fact, only use this connection due to the fact that it is the way of receiving the video streaming from the *drone* and we will be able to command it by using the same USB connection, so we will not need to use anything else in the Remote Controller, only the USB connector. In case we do not need the video streaming we could use only the Remote Controller without the need of connecting a *smartphone* to it.
- **Mobile device:** In the project we are going to use a *smartphone*, however we could connect any mobile device like a tablet as we stated in the illustration 3.6. The *smartphone* used is the Samsung Galaxy S6, so the development of the project will be using the Android API.

Figure 3.9: Device Samsung Galaxy S6 used in the application executions.

3.2.2 Mobile device communication

The DJI SDK is used to provide the smartphone the ability to communicate with the **drone**. The SDK contains several functionalities that we can use to control the **aircraft** and obtain some important telemetry data, for example the status of the battery, sensors, camera control, remote command and more that we are going to use when we implement the communication. The most important parts of the DJI SDK that we are going to need is the aircraft registration in order to grant permission to the application to control the aircraft, the video streaming part and the virtual stick which will be the way to send commands to the aircraft in order to move and rotate.

The first step in the development of the application is registering in the official webpage of DJI and create an application.

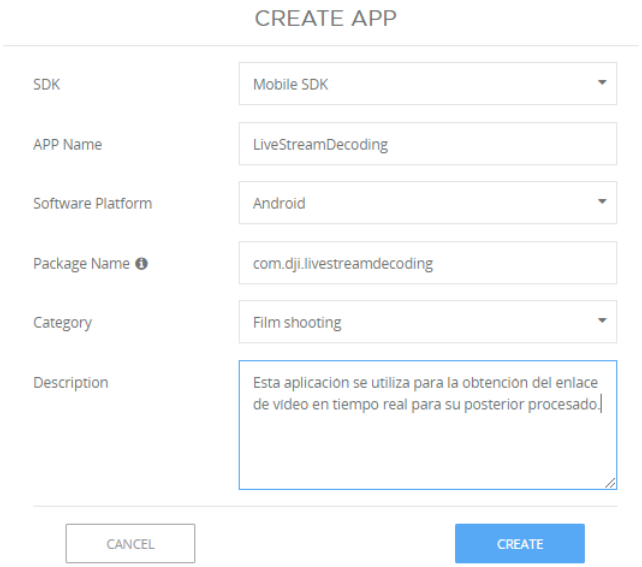


Figure 3.10: DJI SDK Android application creation.

Once this step is completed, we will obtain an "APP KEY" which will be used in order to authenticate our developed application, due to the fact that it will not be able to connect to the aircraft or the dji servers if we do not provide a valid KEY.

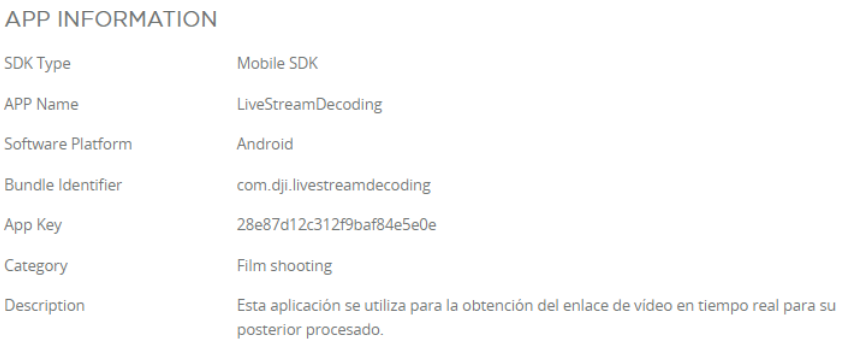


Figure 3.11: Aplicación creada y obtención de App Key

The result window once we have successffully created the application looks as follows, and we will save the APP KEY and proceed with the project creation.

3.2.3 Creating an Android application

The development of the application will be done in Android Studio, which is an IDE used to develop native Android application which provides all the dependencies and libraries needed for the project to run in Android.

Android Studio has one advantage and it is the fact that the Android version system is completely integrated, so we will only have to specify which is the minimum Android version that we want our application to run and it will compile it by using the needed dependencies to provide compatibility across all the selected versions.

The main key feature of using Android Studio is because it provides the tools to download any required library file that is not installed in our system and install and link it automatically so the process is really fast and fail safe.

The required steps to take in order to create a project will be described into the next paragraphs.

In Android Studio we will create a new project selecting the option of Android 5.0 (Lollipop) or higher, due to the fact that this version is the minimum version that is compatible with the DJI SDK. We will leave the other options unchecked because we do not need compatibility with other devices.

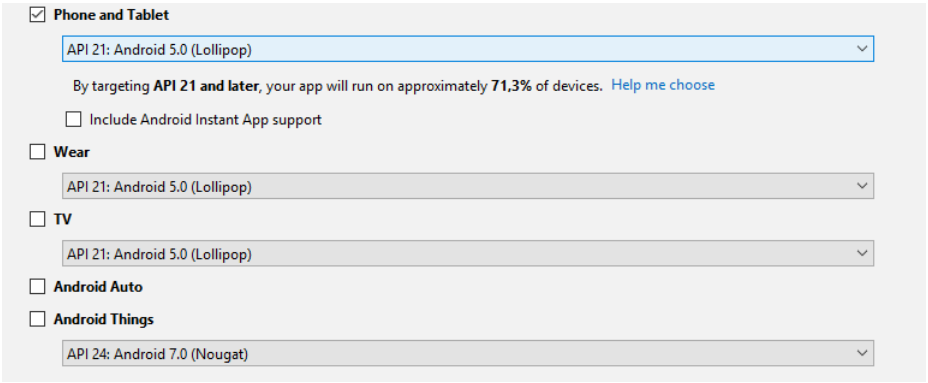


Figure 3.12: Creating the android appplication with 5.0 Android compatbility.

We will select an empty layout, as we will later specify the custom components that we are going to use in each view of the application.

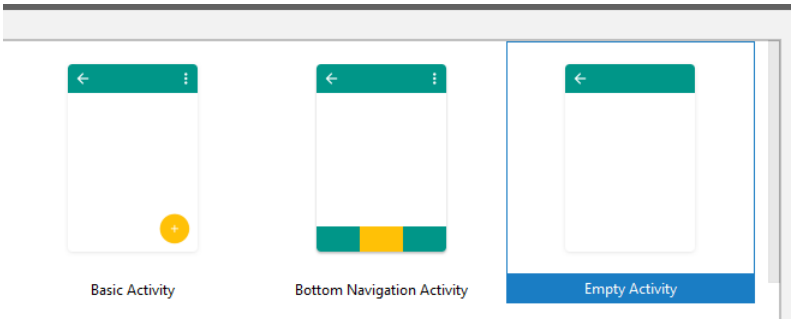


Figure 3.13: Empty Activity selected

Once the project is created we will see a file structure which will be similar to the following:

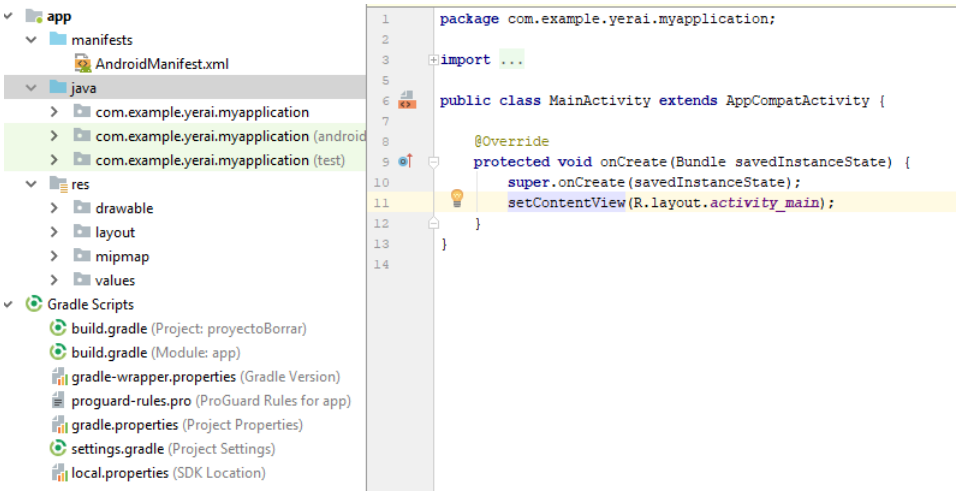


Figure 3.14: Structure of the Android Studio created project

We are going to mention the 3 main parts of the created project.

- AndroidManifest.xml: This file is needed in all the projects, because it provides to the Android System all the needed information about our application. It has some important functionalities: Establish the java package name for the application and enumerate the application components (for example: the activities that we need to define in order to execute them). In the created project, the AndroidManifest.xml file is the following:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
    package="com.example.yerai.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"
     />
                <category android:name="android.intent.category.
    LAUNCHER" />
            </intent-filter>
        </activity>
```

```
18      </application>
19  </manifest>
```

Listing 3.1: AndroidManifest.xml

- Res folder: In this folder we will create all the views that we will use to visualize all the data of the classes of our application, in the views we will define a layout in which we will include some components (buttons, textfields, image frames...) Furthermore, in this folder we will define the list of colours that we are going to need in all the application and some styling parameters.

- build.gradle: In this archive we will define the compilation configuration of our project. It has some key parts as the "buildscript" block in which we will define the repositories and dependencies needed for our project. By default it creates 2 build.gradle files. The first one is located in the project root and the other in the application folder. The one located in the root is used to add the configuration common to all the modules of the project and the low level dependencies, for example the android ones, along with the project build configuration.

```
1  buildscript {
2      repositories {
3          google()
4          jcenter()
5      }
6      dependencies {
7          classpath 'com.android.tools.build:gradle:3.1.3'
8      }
9  }
10
11 allprojects {
12     repositories {
13         google()
14         jcenter()
15     }
16 }
17
18 task clean(type: Delete) {
19     delete rootProject.buildDir
20 }
```

Listing 3.2: Build.gradle root file

On the other hand, the build.gradle located in the application folder we will have a first block that is the "apply plugin" used for gradle to build the project, and it enables the block "android" in which we must specify the minimum Android version that we desire and the target Android SDK version. As we have selected Android 5.0, we should specify a minSDK version 21, even though the target SDK version will be Android 8.0 which is the SDK version 28. In this file we will define the application dependencies, for example the DKI SDK that we will add in the next steps of Gradle configuration.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.example.yerai.myapplication"
        minSdkVersion 21
        targetSdkVersion 28
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0-rc02'
    implementation 'com.android.support.constraint:constraint-
    layout:1.1.3'
}
```

Listing 3.3: Build.gradle de application file

Gradle configuration

As stated in the previous section, the build.gradle file is used to configure the dependencies of our application. We will add the DJI SDK dependency by editing the file and adding the following lines:

```
    packagingOptions{
        doNotStrip "*/*/libdjivideo.so"
        doNotStrip "*/*/libSDKRelativeJNI.so"
        doNotStrip "*/*/libFlyForbid.so"
        doNotStrip "*/*/libduml_vision_bokeh.so"
        doNotStrip "*/*/libyuv2.so"
        doNotStrip "*/*/libGroudStation.so"
        doNotStrip "*/*/libFRCorkscrew.so"
        doNotStrip "*/*/libUpgradeVerify.so"
        doNotStrip "*/*/libFR.so"
        exclude 'META-INF/rxjava.properties'
    }

    dependencies {
        compile fileTree(include: ['*.jar'], dir: 'libs')
        compile 'com.android.support:appcompat-v7:25.3.1'
        compile 'com.android.support:multidex:1.0.1'
        compile 'com.squareup:otto:1.3.8'
        compile ('com.dji:dji-sdk:4.7.1'
        provided 'com.dji:dji-sdk-provided:4.7.1'
    }
```

Listing 3.4: Introduced changes in Gradle.build

Even though we are not directly going to use some of the libraries that we declare, for example the libdum vision or GroundStation, but those are used indirectly by the DJI SDK, so we will need to include them in order to compile the project.

Once this modification has been done, an alert message will show warning us that the file build.gradle has been modified and a synchronization is needed. This is important to mention because it is a really useful feature of Android Studio, due to the fact that syncing the dependencies we will be able to use the custom syntax of the dependencies and the classes of the dependencies added, and the project will compile successfully without errors.

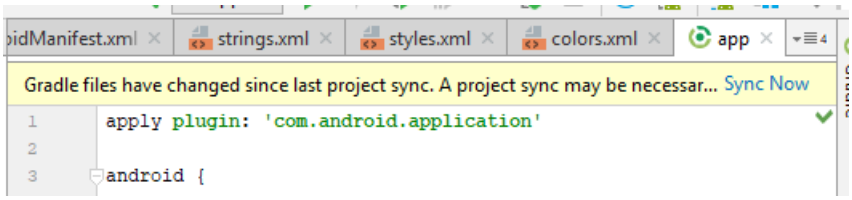


Figure 3.15: Alert prompt showing the change in the build.gradle file and the Sync Now button

The next step of the development is the configuration of the file "AndroidManifest.xml"

Configuración del AndroidManifest.xml

As we have seen before, the AndroidManifest file is really important to the Android system to know every needed information about our application. We should configure it by adding the following sections:

Inside the application tag, we will create a new field called "meta-data" in which we will introduce our APP KEY that we generated in the previous section. This key should have the name "com.dji.sdk.API_KEY" in order to be recognized by the DJI API and inside the tag "value" we will paste the generated KEY.

We will need to add too inside the tag "application", the declaration of the activities "ConnectionActivity" and "MainActivity" which correspond to the classes that we are going to create in the following sections, and will be used by the Android System to know which are our classes and being able to instantiate and show them. The first class will be the one used for registering the SDK on DJI servers and activate it, and will establish the first connection with the drone. The second class called MainActivity will access to the video stream and will show it to the user.

One important step is to add the action "USB ACCESORY ATTACHED", because of the fact that we are going to use the USB port of the mobile device in order to communicate with the drone. By declaring this action, we will generate a call to our code and we will be able to know that this event has happened.

The final code to add is the following:

```
<meta-data
    android:name="com.dji.sdk.API_KEY"
    android:value="d7eeb7484ca2439d95d7f5b3" />
```

```
<activity
    android:name=".ConnectionActivity"
    android:launchMode="singleTop"
    android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
        <action android:name="android.hardware.usb.action.
    USB_ACCESSORY_ATTACHED" />
    </intent-filter>
    <meta-data
        android:name="android.hardware.usb.action.
    USB_ACCESSORY_ATTACHED"
        android:resource="@xml/accessory_filter" />
</activity>
<activity android:name=".MainActivity"
    android:screenOrientation="landscape"
    android:configChanges="orientation|screenSize"></activity>
```

Listing 3.5: Changes in the AndroidManifest.xml file

Application authentication and DJI server connection

Once we have finished editing the gradle.build file, the next step is the creation of the class that will be used to authenticate the application and to validate the drone with the DJI servers.

In order to do this, we will create the class "Connection Activity" which will inherit from the class "Activity" and will implement the method "View.OnClickListener.

We will override the method "onCreate" inherited from the class "Activity", so that when the application creates this view, we will be able to establish the behaviour of the class instead of using the default behaviour of the parent class. In the method onCreate we will call the father method with the statement "super.onCreate(savedInstance)", and we will define the view that we will show to the user and call the method "initUI".

In the method "initUI" we will instantiate and intialize the components defined in the view that we are going to show in the screen, we have some field text and a button to start the application. Furthermore, after loading the view we will call the method "startSDKRegistration".

The function "startSDKRegistration" will contain the statements required to register our application by using the DJI SD. The steps to accomplish this is by calling the method "DJISDKManager.registerApp()" and overriding the method "onRegister", in order to being able to know the registry state, so that in the moment that the app registers successfully we will be able to start our main application.

The reduced final code of this class, without including view element and components (fieltexts and buttons) and other non-important code, is the following:

```
public class ConnectionActivity extends Activity implements View.
    OnClickListener {

```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_connection);
 initUI();
}

private void initUI(){
 //Oculto: Inicializar las vistas y botones...
 startSDKRegistration();
}

private void startSDKRegistration() {
 if (isRegistrationInProgress.compareAndSet(false, true)) {
  DJISDKManager.getInstance().registerApp(
   new DJISDKManager.SDKManagerCallback() {
     @Override
     public void onRegister(DJIError djiError) {
      if (djiError == DJISDKError.REGISTRATION_SUCCESS) {
       DJISDKManager.getInstance().startConnectionToProduct();
       showToast("Register Success");
      } else {
       showToast("Register failed");
      }
      isRegistrationInProgress.set(false);
     }
  });
 }
}

@Override
public void onClick(View v) {
 switch (v.getId()) {
  case R.id.btn_open: {
   Intent intent = new Intent(this, MainActivity.class);
   startActivity(intent);
   break;
  }
  default:
   break;
 }
}
}
```

Listing 3.6: ConnectionActivity.java

As we can see, the last method of the code block above is the function "onClick", which overrides the method of the parent class "Activity". By using this method, we will establish the behaviour of the button "OPEN", and we will set it to open the "MainActivity" class, in which we will be able to acquire the video streaming from the drone.

The class "ConnectionActivity" with its loaded view is the following:

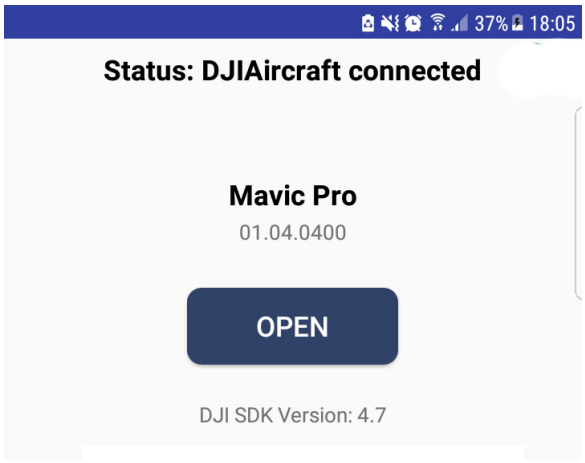


Figure 3.16: Connection Activity user interface

In the picture we can see that we obtain the information about the drone, and show it to the user in the textfields that we declared above, the information displayed is the firmware version of the aircraft and the DJI SDK version that we are using in the application. The OPEN button is enabled when the authentication of the aircraft is done successfully with the method "startSDKReegistration" that we mentioned above. Once we press the button "OPEN" we will start the class "MainActivity" that we will describe in the next section.

Acquire the real time video streaming from the aircraft

The real time video streaming from the drone is acquired by following the next schema:

Figure 3.17: Decoding schema of the real time video link.

As we can see, the first step is getting the video buffer from the drone. To do so, we will create the class "DJIVideoStreamDecoder.java" which will implement the class "NativeHelper.NativeDataListener" from DJI SDK.

This class will contain 5 methods that are the key points in order to obtain the video streaming. The first one is the constructor of the class, which calls the method "startDataHandler" which will be described below, furthermore, it creates a new thread that will be used to obtain the data packets from the drone and creates an object of the class "Handler", and overrides the method "handleMessage"

in order to be able to introduce our own implementation and call the class "NativeHelper.parse" to group and parse the data packets and create the data frames that we need.

The following step is describing the method "startDataHandler" which creates a new thread that will be used to process the received data frames. We will instantiate an object of the class "Handler" and we will override the method "handleMessage" and in this method, the message is already a data frame in H264, so we will be able to send it to decode.

The simplified schema of this created behaviour is the following, we obtain the data buffer from the DJI Mavic PRO and we will send the data packets to the method "NativeHelper.parse()" that will parse the frames and encode them in H264, separating the received data in frames.

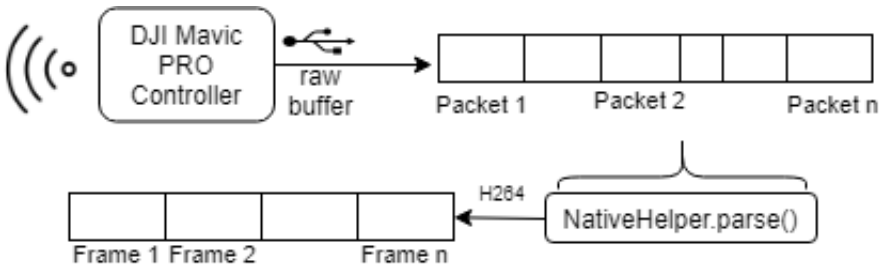


Figure 3.18: Frame decoding process.

The method decodeFrame() is used to access the H264 encoded frames buffer and decode it to introduce all the frames in a new buffer in YUV format inside the class "CodecManager", once we have finished this process we will have a buffer that we will be able to process and use it inside our application, because the YUV format can be easily used by our android application.

Finally, the method "initCodec" creates an object of the class "MediaFormat" in which we will specify the format that we want to decode. We will use the format "YUV420Planar.

The simplified code of this class is the following:

```
1  public class DJIVideoStreamDecoder implements NativeHelper.
       NativeDataListener {
2    private MediaCodec codec;
3    public static final String ENCODING_FORMAT = "video/avc";
4
5    private DJIVideoStreamDecoder() {
6     startDataHandler();
7     handlerThreadNew = new HandlerThread("Parser thread");
8     handlerThreadNew.start();
9     handlerNew = new Handler(new Handler.Callback() {
10      @Override
11      public boolean handleMessage(Message msg) {
12        byte[] buf = (byte[])msg.obj;
13        NativeHelper.getInstance().parse(buf, msg.arg1);
14        return false;
```

```
15        }
16      });
17    }
18
19     private void startDataHandler() {
20      dataHandlerThread = new HandlerThread("frame data handler thread")
        ;
21      dataHandlerThread.start();
22      dataHandler = new Handler(dataHandlerThread.getLooper()) {
23        @Override
24        public void handleMessage(Message msg) {
25             decodeFrame();
26          }
27        }
28      };
29     }
30
31    private void decodeFrame() throws Exception {
32      ByteBuffer buffer = codec.getInputBuffer();
33      buffer.put(inputFrame.videoBuffer);
34      // Feed the frame data to the decoder.
35      codec.queueInputBuffer(inputFrame.size, inputFrame.pts, 0);
36      // Decode to YUV.
37        ByteBuffer yuvDataBuf = codec.getOutputBuffer(outIndex);
38        yuvDataListener.onYuvDataReceived(yuvDataBuf, width, height);
39    }
40
41
42    public void parse(byte[] buf, int size) {
43      Message message =handlerNew.obtainMessage();
44      message.obj = buf;
45      message.arg1 = size;
46      handlerNew.sendMessage(message);
47    }
48
49    private void initCodec() {
50      MediaFormat format = MediaFormat.createVideoFormat(ENCODING_FORMAT
        );
51      format.setInteger(MediaFormat.KEY_COLOR_FORMAT,
        COLOR_FormatYUV420Planar);
52      codec = MediaCodec.createDecoderByType(ENCODING_FORMAT);
53    }
54  }
```

Listing 3.7: DJIVideoDecoder.java

Drawing the video on screen

In order to draw the realtime video from the drone on the user's screen we will create the class "MainActivity.java". This class also inherits from "Activity" class and will implement the method "DJICodecManager.YuvDataCallback".

We will do the same as we did with the class "ConnectionActivity", so we will override the method "onCreate" of the parent class "Activity", in order to load the required view and initialize the class and its components.

The most important object of this class is the instance of the class DJICodecManager. As we saw in the previous section, we make use of a buffer that is going to be filled with the received frames in YUV420Planar format. We will access to this buffer specifying that we want to access the frames in YUV format and passing the callback function that is going to be called whenever a frame is ready in the buffer. The way of achieving this objective is by making our class "MainActivity" to implement the method DJICodecManager.YuvDataCallback as we said before.

Due to this implementation, we need to override the method "onYuvDataReceived", which will be the function where we will receive the frames once we set the callback of DJICodecManager to the actual instance of the class "MainActivity". We will be using this method to send the frame to the process video class or create a new thread and show the image on the user screen.

Code snippet of the class:

```
public class MainActivity extends Activity implements DJICodecManager.
    YuvDataCallback {
  private static byte[] yuvBytes;
  private DJICodecManager mCodecManager;
  private int width = 1280;
  private int height = 720;
  private Imageview imageViewTest;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    imageViewTest = findViewByID(R.layout.image_view_test);
    enableYUVData();
  }

  private void enableYUV(){
     mCodecManager = new DJICodecManager(getContext(), width, height);
     mCodecManager.enabledYuvData(true);
     mCodecManager.setYuvDataCallback(this);
  }

  @Override
  public void onYuvDataReceived(ByteBuffer yuvFrame, int width, int
    height) {
    byte[] bytes = new byte[dataSize];
    yuvFrame.get(yuvbytes);
    if(DJIDetectorActivity.initialized) {
      DJIDetectorActivity.onFrameReady(yuvBytes);
    } else {
      Runnable r = () -> {
        int[] rgbBytes = convertYUV420_NV21toRGB8888(yuvbytes, width,
    height);
        bmp = createBitmap(rgbBytes, width, hight, Bitmap.ARGB_8888);
        if(bmp != null) imageViewTest.setImageBitmap(bmp);
        imageViewTest.postInvalidate();
      };
    }
```

```
36  }
```

Listing 3.8: MainActivity.java

Once this class is created, we can compile the project and run it. In the view we can see that we are showing properly the frames in the screen. The result of the view can be seen in the appendix, section A

3.3 Neural Network training process

Once we completed all the previous steps, we can start the process of training the Neural Network. The chosen model, as we mentioned early, is the Single Shot MultiBox Detector (SSD).

There are three main steps to carry out on the training process: dataset creation, the training step and result evaluation. We are going to follow this steps over the next sections.

3.3.1 Dataset creation

To begin with the dataset creation we need to download several images. Firstly a search on Google Images was performed, in which 500 images of 3 different classes (pertaining to DJI Phantom, DJI Inspire and DJI Mavic PRO drones) were downloaded.

Figure 3.19: Left: DJI Phantom, Center: DJI Mavic, Right: DJI Inspire

Once the initial download process was completed, the Python script "labelImg" was used in order to tag the pictures. We can obtain this script in the repository https://github.com/tzutalin/labelImg.

The instructions of the program are pretty straightforward, we navigate to the folder where we stored our downloaded images and execute the script by using the command "python3 labelImg.py".

When we start the application we can observe that the user interface is really intuitive and lets us create different regions and classify depending on which object falls in the created region.

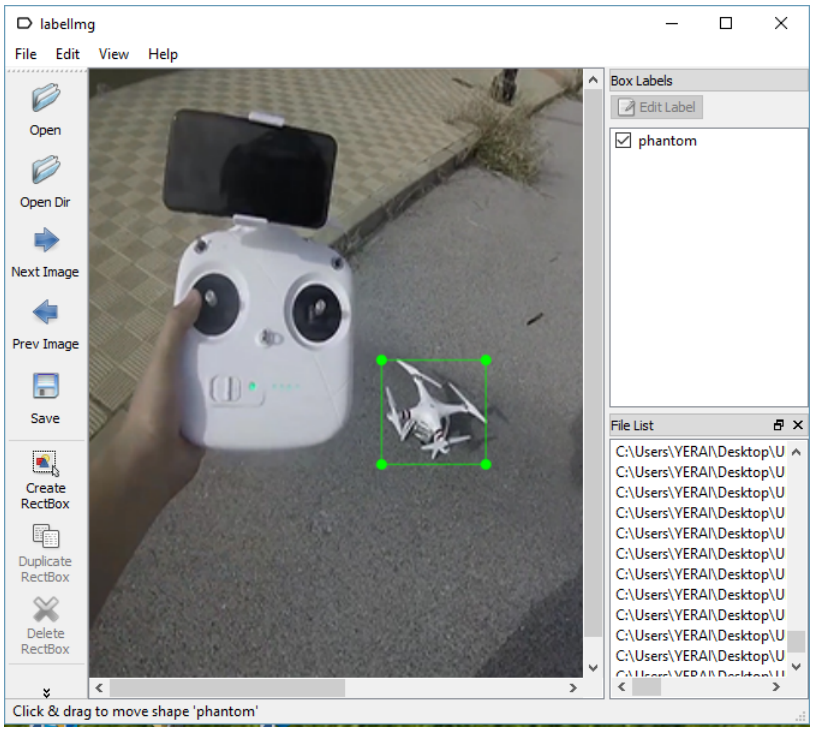


Figure 3.20: LabelImg application

Then, after the labeling process is finished, a XML file will have been created per each one of the image labeled. This file will have a content similar to the following:

```
<annotation>
  <folder>dataset</folder>
  <filename>mavicPro00898.jpg</filename>
  <path>dataset_to_label/mavicPro00898.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>480</width>
    <height>320</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>mavicPro</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>204</xmin>
```

```
      <ymin>139</ymin>
      <xmax>294</xmax>
      <ymax>215</ymax>
    </bndbox>
  </object>
</annotation>
```

Listing 3.9: Content of the generated XML file

The next stage, is generating a CSV file so that our network can train with the created dataset, where we will have the content of all the generated XML files on one single file where each entry will represent a labeled object in a determined image.

This process is achieved by using the next piece of code, which iterates over the directory whilst filling the data from the XML files on a CSV file.

```
import os
import xml.etree.ElementTree as ET

def xml_to_csv(path):
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            if(member[0].text=='phantom' or member[0].text=='mavicPro'
    ):
                filename = root.find('filename').text
                value = (filename, find('size')[0], find('size')[1])
                xml_list.append(value)
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df

def main():
  for directory in ['train','test']:
    image_path = os.path.join(os.getcwd(), 'images/{}'.format(
    directory))
    xml_df = xml_to_csv(image_path)
    xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None)
    print('Successfully converted xml to csv.')

main()
```

Listing 3.10: Code to perform the CSV generation from XML

The CSV file generated file will be similar to the next example, as we said previously, it organizes all the created XML files, in each entry we will see the coordinates of a region that delimitates the object present on a determined picture.

```
filename,width,height,class,xmin,ymin,xmax,ymax
test/mavicPro00898.jpg,480,320,mavicPro,204,139,294,215
test/mavicPro00902.jpg,480,320,mavicPro,243,69,328,159
test/mavicPro00903.jpg,480,320,mavicPro,242,83,341,156
test/mavicPro00926.jpg,480,320,mavicPro,223,62,308,142
test/mavicPro00930.jpg,480,320,mavicPro,194,89,330,191
test/mavicPro00931.jpg,480,320,mavicPro,191,79,343,188
```

Listing 3.11: Generated CSV file.

Finally, the last step needed is generating a tfrecords file. This is needed because it is the format in which the Tensorflow detection API process the data. It is a binary file in which we will have a series of entries (records) per each dataset image.

The generation of these files (we need two, one for the training data and one for the validation data) is performed by using the code generate_tfrecord.py which was obtained in this github repository.

We need to execute the script with the next commands:

```
$> python generate_tfrecord.py --csv_input=data\train_labels.csv --
    image_dir=images\train --output_path=train.record
$> python generate_tfrecord.py --csv_input=data\test_labels.csv --
    image_dir=images\test --output_path=test.record
```

Listing 3.12: Tfrecords generation commands.

Once executed, two binary files will be created named test.record and train.record. We will proceed with the configuration of the model to be trained.

3.3.2 Neural Network model configuration

So as to train the SSD Neural Network model by using the Tensorflow API, we will need to provide the desired configuration, in which we will specify some important parameters as:

- Number of classes in our dataset.
- Neural Network structure.
- Path of our tfrecords file.
- Path to our dataset, which contains the images present on the tfrecord file.

For further understanding, the configuration file used can be viewed on the Appendix B.

Once the configuration process is finished, we will start the training of the Network. We will create a folder called "train" which is where all the checkpoints and the saved model with its weights will be saved.

Finally, the last step to start the training process, is start it by using the following command: "python train.py –train_dir=train".

We will analyze the training process in the following section.

3.3.3 Training analysis

Owing to monitor the training process, when we installed the dependencies we also included the Tensorboard library.

This library gets a huge importance with this purpose, we will start the tool by specifying which is the folder where the training logs are being saved, in our example as we have established the path "train" we will make use of the command "tensorboard –logdir="train".

Once the program starts, it will generate a IP adress which will be introduced on the web browser and we will be prompted with the following workspace:

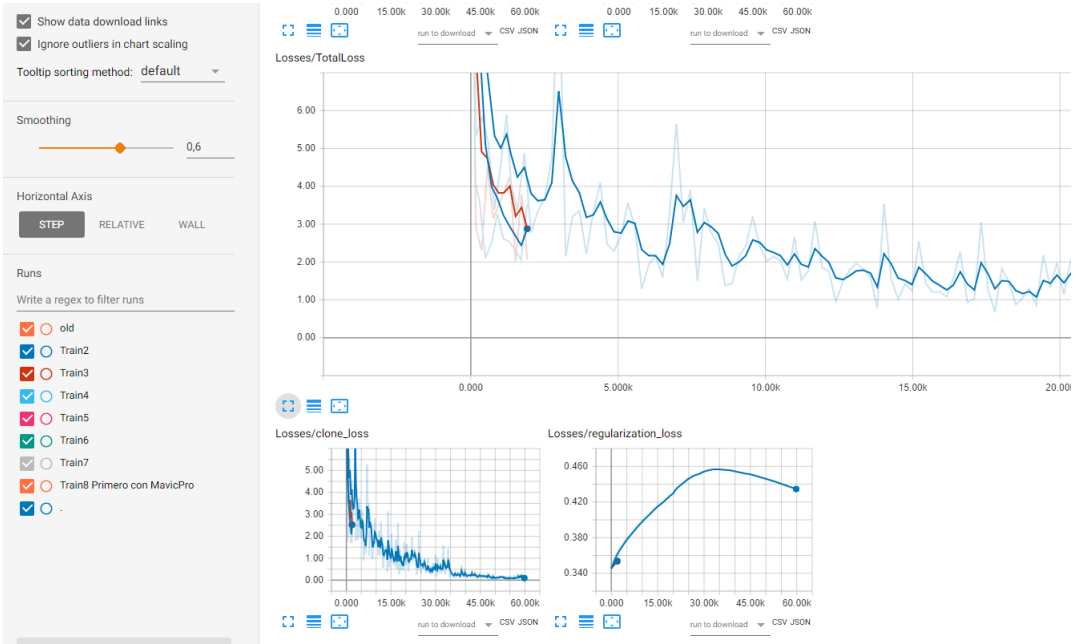


Figure 3.21: Tensorboard panel

In this user interface we have several metrics that we can observe, but firstly we will configure the smooth parameter to 0, so that we do not get that smooth generated graphs.

The set of charts we can visualize is really large. One of the most important for our application is the "Loss" (45) chart, by using this chart we can monitor and determine when to stop the training process.

After the first training process was completed we obtained the following result:

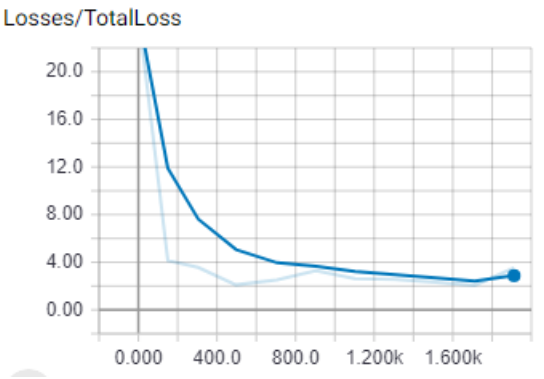


Figure 3.22: First training process chart.

As we can observe, the loss value is decreasing considerably over the first 400 epochs, however from this point on, it is established around 3.5 and 4 which is a really high value for the loss function.

Therefore, the obtained results are not very satisfactory. After analyzing the data and examining the documentation associated to the training process there was some information related to the data augmentation process, specifically the crop parameter, which cuts the image and generates a new one in order to enlarge the dataset.

After consulting the related information to that cropping function, it was a possible cause of the issue. As a recommendation in some articles, the suggestion was to leave only activated the rotation option, which will rotate the image in random angles owing to generate new data during the training process.

Once this option was disabled and the training process was started again, we obtained the following results:

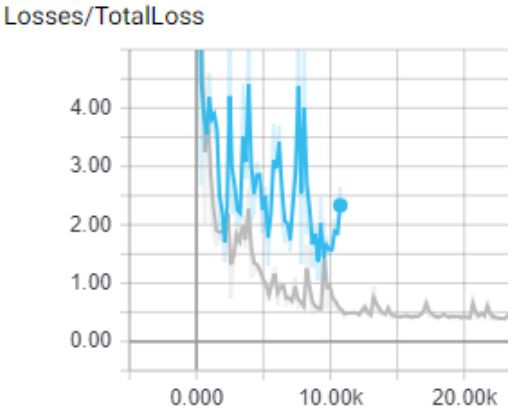


Figure 3.23: Second training metrics after eliminating the cropping parameter.

As we can observe, the blue lines correspond to the training process with the cropping parameter activated, and the gray lines correspond with the option disabled. We have obtained a loss value near to 0.5, which is relatively good considering the attributes of the SSD Neural Network.

3.3.4 Neural Network outcome improvements: Method 1

When initializing the model and executing it over a set of images that were not seen before, we obtained a results that are not good enough, but for example in the pictures where the drone to be detected is near to the camera and it is big enough, the results obtained are really good.

The next step in the developing process was augmenting the dataset, however the procedure followed now is different. After analyzing the videos of the drones that were supposed to be detected, an important fact could be extracted.

The images obtained from Google Images where pretty similar, a lot of them were taken for selling the drone and the positioning of the camera was identical to other pictures. The environment where the picture was taken focusing on this idea, and sometimes it did not represent a real working environment.

The process followed was downloading a series of videos that showed the drone operating in an everyday use environment, which included a lot of different takes with a large variety of sizes, rotations and backgrounds.

The videos were downloaded, and the images were extracted from them, extracting pictures at a rate of 3 frames per second, so each second of video we will obtain 3 images.

Thanks to this step, the obtained dataset with a 5 minutes videos was approximately of 1000 images. The fact of saving 3 frame per second is because normally the drone movements are pretty fast, so in the time elapsed in each frame, 333 milliseconds, is sufficient to the drone to slightly change its orientation and scale so the physical characteristics of the aircraft may vary significantly.

With this method the following results were obtained:

Figure 3.24: Results obtained from the training process by using the augmented dataset method by extracing frames from a video.

We can analyze the results and observe a huge difference between the first trained model and the model trained with the images extracted from videos. The results at short and medium distance are pretty good, although the far distance results are not good enough. Even though we need to remember that detecting small objects like drones is really difficult, due to the fact that if we increase a bit the distance of the camera, the pixels representing the drone in the image are certainly reduced and we are not able to extract a lot of characteristics due to this limitation.

3.3.5 Neural Network outcome improvements: Method 2

Seeing that the obtained results could be improved, there was another method to test with the purpose of improving the Neural Network outcome.

The hold procedure was as follows:

- Create a new dataset using the method 1, with different images to the ones that are already in the dataset.
- Centering our efforts on the images that the Neural Network fails to classify without neglecting the others. For instance, as the model failed to detect objects in the medium-far distance, more images of the drones under this circumstances were included, as the previous dataset did not include enough images of this type.
- Execute the model trained with the Method 1 and classify the images of this new dataset.

With this last step, we obtained some interesting results:

- The new dataset was formed of 2000 new images, 4/10 parts of these images, the classification performed was correct and the confidence value was superior to 90%.
- In 2.5/10 parts of these set of images, the Neural Network provided a positive result but its confidence value was inferior to 60%.
- The 3.5/10 parts left, the Neural Network failed to detect or detected a false region in the image.

Once this data was analyzed, the following actions were taken:

- A 10% of pictures of the first group were introduced randomly in our dataset. (The other 90% was discarded).
- A 50% of images of the second group were introduced randomly in our dataset, where the model result was correct but the confidence was %60.
- All the images (100%) of the images were the network failed were introduced into our dataset.

By using this method, using the neural network to classify and tag, we automatically processed the whole dataset and label the images of the first and second group (90% and 60% confidence values), due to the fact those images were classified properly. The only required effort was labeling the images where the Neural Network failed completely (third group).

After training the model again with the new dataset that included the new images, the result obtained was really good and were pretty much the ones that were desired, keeping in mind that the Neural Network were able to detect the 3 types of drones that were planned, and considering the fact that the detection distance is really difficult to increase due to the fact that drones' sizes are really reduced. Overall, the detection distance is approximately 3 meters.

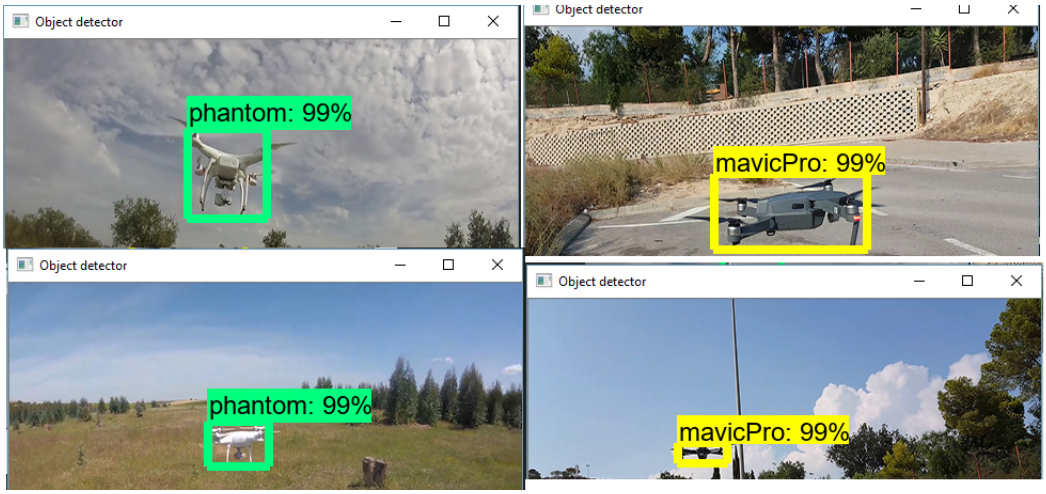


Figure 3.25: New results of the training process with the method 2.

3.4 Running Neural Networks on Android

The next step in the project is running the trained model in an Android device. To accomplish this goal, we are going to configure the Tensorflow library in the device.

We will continue with the project created on the section 3.2.3, we will add some changes that will be needed for this purpose.

The first step is adding some parameters in the second line of the file "build.gradle" which should be inside the directive "dependences". The result of this addition is the following:

```
    dependencies {
        compile 'org.tensorflow:tensorflow-android:+'
        compile fileTree(include: ['*.jar'], dir: 'libs')
        compile 'com.android.support:appcompat-v7:25.3.1'
        compile 'com.android.support:multidex:1.0.1'
        compile 'com.squareup:otto:1.3.8'
        compile ('com.dji:dji-sdk:4.7.1'
        provided 'com.dji:dji-sdk-provided:4.7.1'
    }
```

Listing 3.13: Code to be added to the build.gradle file in order to use tensorflow.

The next change is inside the file "ANdroidManifest.xml" in which we should define a new activity which will be called "DJIDetectorActivity", defining it as follows:

```
<activity android:name="com.dji.streamdecoding.tensorflow.
    DJIDetectorActivity"
  android:screenOrientation="portrait"
  android:label="@string/activity_name_dji_detector">
  <intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
    <category android:name="android.intent.category.LEANBACK/">
  </intent-filter>
</activity>
```

Listing 3.14: Code to be added in the AndroidManifest.xml file

Once this step is completed, we can proceed with the creation of the class "DJIDetectorActivity.java", which will extend from the class "Activity".

In the application we are going to compare the models "Tiny-Yolo" and "SSD-Mobilenet" so we will create a button to switch between them.

As always, in the class we will override the method "onCreate" from the Activity class, in which the components of the view and the new button to create will be defined.

In addition, we will create a new thread that will serve us to show the image and process it, the code added until this moment is the following:

```
public class DJIDetectorActivity extends Activity {
    Button changeDetector;

```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //...Initialization of the components...//
        new Thread(() -> {
            try {
                while(true) {
                    Thread.sleep(33);
                    processImage();
                    refreshView();
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }).start();
    }
```

Listing 3.15: DJIDetectorActivity.java

As we can see, we invoked 2 functions. One of them is "refreshView()" which we delegate to show the video on the device screen and show the detections.

Those detections are added in a separate view, so what we really do is paint the video frame and create an overlay with the detections above it.

The code to be added is the following:

```
public static void onFrameReady(byte[] yuvBytes){
  int[] rgbBytes = convertYUV420_NV21toRGB8888(yuvbytes, width, height
    );
  bmp = createBitmap(rgbBytes, width, hight, Bitmap.ARGB_8888);
}

public void refreshView(){
    runOnUiThread(new Runnable () -> {
        if(bmp != null) ivDetector.setImageBitmap(bmp);
        ivDetector.postInvalidate();
    };);
}
```

Listing 3.16: Method onFrameReady of the class DJIDetectorActivity

In the method "refreshView", we run the UI thread, which corresponds to the user interface, also we call the method postInvalidate() in order to refresh the view when the new picture is available to show.

Furthermore, the function "onFrameReady" was created, and it is called from the class "MainActivity.java", defined in the section 3.2.3, whenever a new frame is ready to be shown and processed.

The last method is called "processImage", in which we will create a new thread to process the image and call the Tensorflow detection API. Moreover, we will use this method to paint the detections in the view that are going to be drawn over the image.

```
protected void processImage() {
 new Runnable() {
 @Override
 public void run() {
```

```
5     List<Recognition> results = detector.recognizeImage(bmp);
6     copyBitmap = Bitmap.createBitmap(bmp);
7     final Canvas canvas = new Canvas(copyBitmap);
8     final Paint paint = new Paint();
9     paint.setColor(Color.RED);
10    paint.setStyle(Style.STROKE);
11    paint.setStrokeWidth(2.0f);
12
13    float minimumConfidence = 0;
14    switch (MODE) {
15      case SSD_MOBILENET:
16      minimumConfidence = MINIMUM_CONFIDENCE_SSD_MOBILENET;
17      break;
18      case YOLO:
19      minimumConfidence = MINIMUM_CONFIDENCE_YOLO;
20      break;
21    }
22
23    List<Recognition> mappedRecognitions = new LinkedList<>();
24
25    for (final Classifier.Recognition result : results) {
26      final RectF location = result.getLocation();
27      if (result.getConfidence() >= minimumConfidence) {
28      canvas.drawRect(location, paint);
29      result.setLocation(location);
30      mappedRecognitions.add(result);
31      }
32    }
33    ivOverlay.paint(canvas);
34   });
35 }
```

Listing 3.17: method processImage of the class DJIDetectorActivity

In the code we can see the method we use to call the "Tensorflow Detection API", we obtain a list of detected objects, that will be iterated and analyzed to see if the confidence value is greater than a fixed threshold. If the threshold policy is satisfied, the detection will be drawn.

The code has been included in the appendix, section C, omitting data related with the views, components and other functions that are not relevant.

The result of this class and the view showing the detection of a drone DJI Mavic PRO in a picture, can be seen also in the appendix, section E.

3.5 UAV command with detections of the Neural Network

The last step of the implementation is using the Neural Network detections to command the aircraft by sending update commands using the DJI SDK.

In order to achieve this goal, we will create a new class called "CommandAircraft" which will be used to receive the detections from the Neural Network and calculate the command needed to be sent.

3.5.1 Axis of an aircraft

In order to command the movement of the aircraft we need to know which axis we need to command to perform the desired movement.

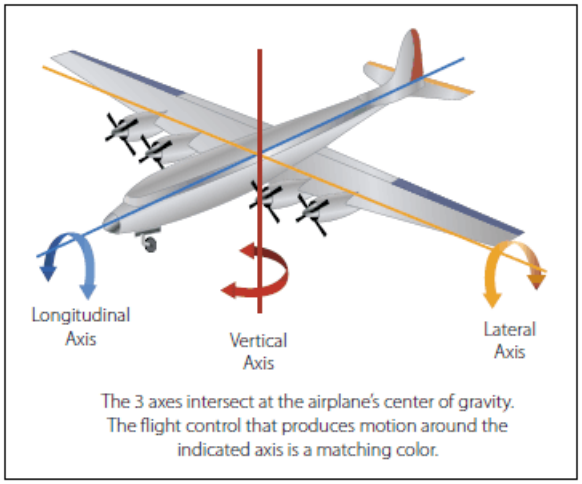


Figure 3.26: Axis of a plane

The 3 axis correspond to the following notation:

- Logitudinal Axis: Roll Axis.
- Vertical Axis: Yaw Axis.
- Lateral Axis: Pitch Axis.

For safety reasons and in order to simplify the algorithm we are only going to send yaw and pitch commands, so we will not modify the altitude or the roll of the aircraft.

By modifying the pitch rotation we will make the aircraft to move forwards or backwards, and the yaw rotation will make the aircraft to change its heading direction by rotating to the left or right.

3.5.2 Yaw command

The calculation for the yaw command is made by taking the center of the image as the 0 and calculating the distance between the center of the image and the center of the detection. If the detection falls in the left part of the image, that distance will be negative and positive if it falls in the right part of the image.

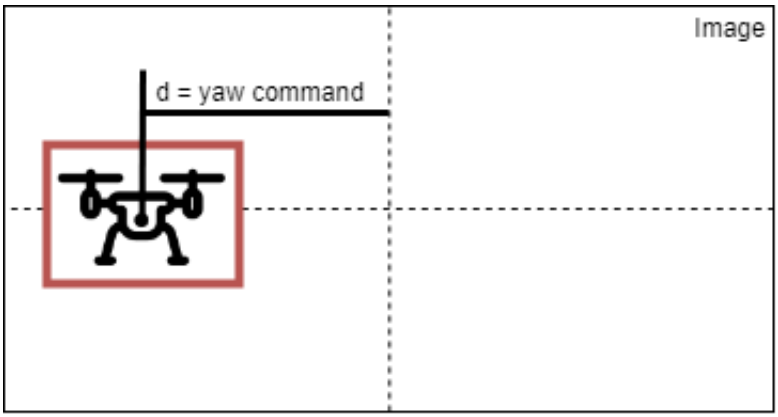


Figure 3.27: Tracking algorithm yaw

With this illustration and the explanation above, we obtain the formula of the yaw command:

$$YawCommand = boxCenter_x - \frac{imageWidth}{2}$$

With this method we will send the aircraft the error compensation which is needed to be taken, being proportional to how far the detection is from the middle point of the image.

3.5.3 Pitch command

With the pitch we will make the aircraft move forwards or backwards, depending on the size of the detection. If the object detected is too big, we will command the aircraft to go backwards (negative pitch) and if the object is too small we will command to go forward (positive pitch).

The way of doing this is once we have the width of the detection we set the desired size of the object to be a third of the image width. So if our image width is 300 pixels, we will try to have the object width of 100 pixels. So if the object is too large, we will send a negative pitch command proportional to its width.

The formula is the following:

$$PitchCommand = \frac{imageWidth}{3} - detectionWidth$$

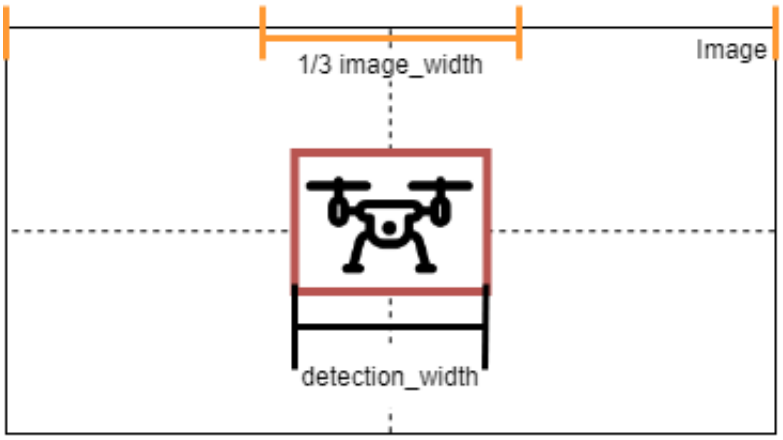


Figure 3.28: Tracking algorithm pitch

By using this calculation, the command will be sent according to the proportional size of the detected object.

3.5.4 Implementing the class CommandAircraft

In order to command the aircraft we will need to declare some important variables, which are a Timer, a SendVirtualStickDataTask and a FlightController.

- The Timer class will be used to send the command repeatedly at a specified frequency.
- SendVirtualStickDataTask is a class which we will create that extends from "TimerTask" and implements the run method and sends the command.
- FlightController is the DJI class that controls the aircraft, which we will need to use to establish the command needed to be sent by providing it the "SendVirtualStickDataTask" task to be run.

```
1  public class CommandAircraft {
2    private float pitch = 0f;
3    private float roll = 0f;
4    private float yaw = 0f;
5    private float throttle = 0f;
6    private int image_width = 1280;
7    private int image_height = 720;
8
9    private Timer sendVirtualStickDataTimer;
10   private SendVirtualStickDataTask sendVirtualStickDataTask;
11   private FlightController fContr;
12
13   private class SendVirtualStickDataTask extends TimerTask {
```

```
14      @Override
15      public void run() {
16        if ( fContr != null) {
17          fContr.sendVirtualStickFlightControlData(
18            new FlightControlData(pitch,roll,yaw,hrottle);
19        }
20      }
21    }
22  }
```

Listing 3.18: Variable declaration and SendVirtualStickDataTask class

The method "init" will be the initializer of the class, which will get the FlightController instance from the DJI SDK, and create the instance of the objects sendVirtualStickDataTask and sendVirtualStickDataTimer.

An important step is to set the VerticalControlMode and RollPitchControlMode to "VELOCITY and YawControlMode to "ANGULAR_VELOCITY", as we will be commanding a velocity to the aircraft, There are two modes to command the aircraft, the first one is "POSITION" which is a relative position from the aircraft, but it is not useful for our purpose because we will need to know the exact position of the object to track from the aircraft.

The second mode is called velocity mode, which accomplish the requirement of following and object because we will set the velocity in yaw according to the algorithm above, and the aircraft will turn in yaw (clockwise or anti-clockwise, depending on the sign of the number) until the algorithm output is 0, so the detected object is in the middle of the picture. The same happens with the pitch, the aircraft will rotate at some speed determined by the output of the algorithm until the width of the detected object is the desired.

```
1  public void init(){
2    fController = ((Aircraft) DJISDKManager.getInstance()
3      .getProduct()).getFlightController();
4
5    sendVirtualStickDataTask = new SendVirtualStickDataTask();
6    sendVirtualStickDataTimer = new Timer();
7    sendVirtualStickDataTimer.schedule(sendVirtualStickDataTask, 200);
8    fContr.setVerticalControlMode(VerticalControlMode.VELOCITY);
9    fContr.setYawControlMode(YawControlMode.ANGULAR_VELOCITY);
10   fContr.setRollPitchControlMode(RollPitchControlMode.VELOCITY);
11 }
```

Listing 3.19: Method init

Furthermore, we will have a method "setCommand" which we will be calling from the class DJIDetectorActivity.java (declared on section 3.4). We will pass in the parameter the Rectangle of the detection and we will calculate the pitch and yaw according to the formula in the sections 3.5.2 and 3.5.3.

```
public void setCommand(RectF location){
  float boxWidth = location.right - location.left;
  pitch = (image_width/3) - boxWidth;

  float boxCenter = location.left + (boxWidth/2);
  yaw = boxCenter - (image_width/2);
}
```

Listing 3.20: Method setCommand()

The complete code of the class can be seen in the appendix section D.

Chapter 4

Result analysis and conclusions

To conclude with the project, we will evaluate the results obtained from the execution of two Neural Network models and elaborate a comparison between them.

And last but not least, we will end by analyzing the studied concepts in addition to extracting the conclusions of the carried out work.

4.1 Evaluation of results

To summarize, we will perform an analysis of the obtained results related to the execution of Neural Network, comparing the results obtained on a computer and the mobile device. This comparison provides a lot of information regarding to the capacity of executing complex computing models in reduced size devices.

The comparison will be performed between the two tested models, the "Tiny-YOLO" and "SSD-Mobilenet".

The first result analyzed is in the laptop were the training process was performed, which includes a Graphics Card Nvidia GTX970M. The data analyzed is the required time to process each frame over 30 frames.

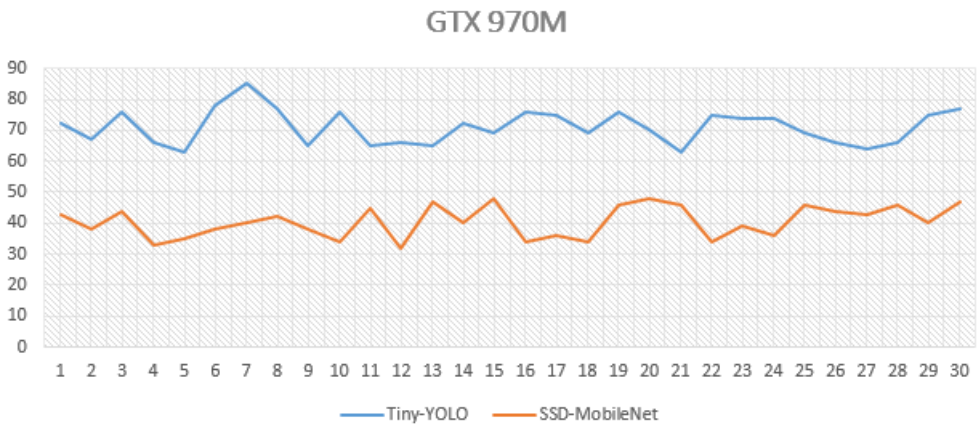


Figure 4.1: Tiny-YOLO and SSD model comparison on a GTX 970M

The conclusions obtained are really good in both scenarios, keeping in mind that the accuracy of the model "Tiny-YOLO" is slightly better than the obtained with the "SSD-Mobilenet". That is an advantage and a disadvantage of both models.

With this data, the amount of frames per second (FPS) that can be processed is the following:

- Tiny-YOLO: 14 FPS.
- SSD-MobileNet: 24 FPS.

The results obtained when the analysis in the mobile device was carried out differ somewhat in the conclusion obtained for the laptop:

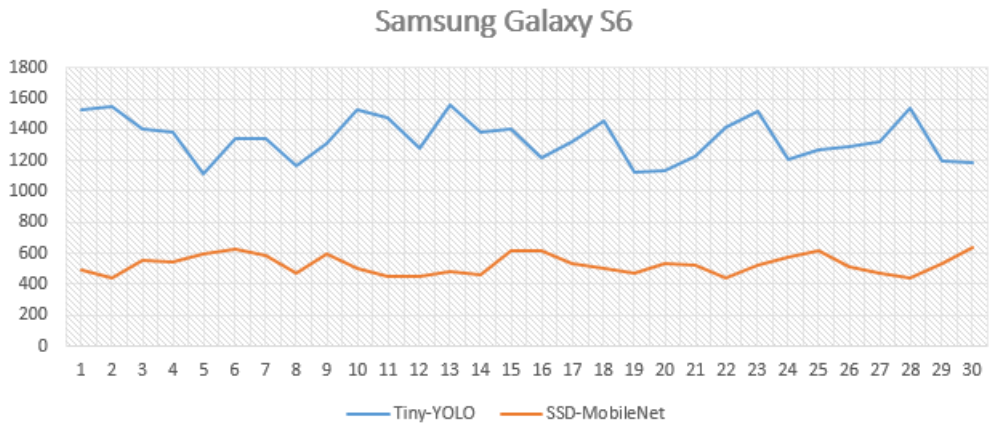


Figure 4.2: Tiny-YOLO and SSD model comparison on a Samsung Galaxy S6

The obtained results are more or less similar, but there are some differences. The model "Tiny-YOLO" experiments a high variability in the required processing time, this might be due to the resource limitation and the optimizations that the Tensorflow framework has included and these optimizations might have affected both models in a different way,

In this scenario, the use of the model SSD-Mobilenet is easily advocated, due to the variability in the performance and the huge difference in the processing time.

The processing throughput in frames per second (FPS) is the following:

- Tiny-YOLO: 0.7 FPS.
- SSD-MobileNet: 1.8 FPS.

At a first glance, the obtained results might be a little discouraging, however we need to keep in mind the limitations of these devices which is caused by the reduced size of them, thus considering the computing capability required to execute a classification Neural Network, the results are pretty decent.

4.2 Final conclusions

To conclude with the project, we need establish an analysis of the key concept of the project.

The developing process started by acquiring the real time video streaming from the Aircraft, and the results obtained were satisfactory.

Some complications arose when trying to run the SSD MobileNet Neural Network due to the lack of computing capabilities of the mobile device, however the obtained results were quite pleasant as we could see in the previous section. The result analysis threw a frame rate of 2 FPS approximately, although this is not ideal, considering the computing requirements of the Neural Networks and the reduced performance of mobile devices, the conclusion we can extract is quite pleasant.

Last but not least, the Aircraft commanding was the final feature of the project and had to be run along with the real time video streaming reception and the SSD MobileNet model. With the obtained output from the Neural Network we could only be able to send two commands to the drone per second, that means a command each half a second. Considering that a drone is small and fast, this data is not really satisfying because in that half second the drone can drastically change its distance and the Neural Network might not be able to recognize it again.

Although, it was the expected result and is quite rewarding, acknowledging once again that the performance under normal conditions and considering that the computing cost of this application and the reduced size of the mobile devices which implies a reduced performance of them.

The final conclusion will be centered on the performed study around the used technology. In this project two concepts that are usually analyzed in a separate way have been treated together.

On the one hand, the complex Neural Network models, that have always required a huge amount of computing resources, have always been analyzed, in the vast majority of the cases, in high end systems. This analysis is completely valid and well-founded if compared with other models under the same conditions.

However, there are some situations in which there is not a clear comparison between the different models and their performance in lower performance systems, so, the use of both models of Neural Networks that have been analyzed in the project is really convenient to make a comparison in conditions where hardware resources are limited.

The project development has implied an important challenge, due to the fact that it was needed to analyze the state of the art of the current computing models, to perform a comparison between some of the models that augured a better performance when they were subjected to low-income condition systems.

Several conclusions can be clearly extracted from this analysis:

- The already traveled path in the Neural Network field has currently experienced a very accelerated growth in performance, in both devices with high or low computational capacity.
- This increase of the yield has caused to be able to experiment with these computational models in devices in which, time back, we could not contemplate. A clear example of this is the use of Neural Networks in mobile devices.
- It is clearly expected that this performance will increase even faster, and we will gradually begin to see even more progress in the models designed for reduced performance systems.

References

[1] C. Larman *Agile & Iterative Development: A Manager's Guide.* AddisonWesley, 2004

[2] L. Rising and N. S. Janoff *The Scrum software development process for small teams* IEEE Software, 2000

[3] K. Beck *Extreme Programming explained: Embrace change.* Addison-Wesley, 2004

[4] Eric Brechner *Agile Project Management with Kanban* Microsoft Team, 2015

[5] Paul Viola Michael, Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features.* https://www.cs.cmu.edu/ẽfros/courses/LBMV07/Papers/viola-cvpr-01.pdf

[6] Navneet Dalal, Bill Triggs. *Histograms of Oriented Gradients for Human Detection.* https://ieeexplore.ieee.org/document/1467360/

[7] Artem Rozantsev, Vincent Lepetit, Pascal Fua. *Flying Objects Detection from a Single Moving Camera.* CoRR. 2014

[8] Dr Simone Teufel, Prof Ann Copestake. *Dataset Splitting* Cambridge University. Machine Learning and Real-world Data. https://www.cl.cam.ac.uk/teaching/1617/MLRD/handbook/dataset-splits.pdf

[9] Wen, Z. and Li, B. and Kotagiri, R. and Chen, J. and Chen, Y. and Zhang, R. *Improving Efficiency of SVM k-fold Cross-validation by Alpha Seeding.* ArXiv:1611.07659, 2016. http://arxiv.org/abs/1611.07659

[10] Jeff Schneider *Cross Validation* Carnegie Mellon University https://www.cs.cmu.edu/ schneide/tut5/node42.html

[11] M. Mishra and M. Srivastava. *A view of Artificial Neural Network.* ICAETR. 2014

[12] Jürgen Schmidhuber. *Deep Learning in Neural Networks: An Overview* CoRR. 2014

[13] Koushik, Jayanth. *Understanding Convolutional Neural Networks* ARXIV. 2016

[14] Ilya Loshchilov, Frank Hutter. *Fixing Weight Decay Regularization in Adam* dblp. 2017

[15] ujjwalkarn. *An Intuitive Explanation of Convolutional Neural Networks* 2016. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[16] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, Mohammed Bennamoun. *A Guide to Convolutional Neural Networks for Computer Vision* Morgan and Claypool Publishers. 2018. [p48].

[17] Andrew Ng. *Convolutional Neural Networks* https://es.coursera.org/lecture/convolutional-neural-networks/padding-o7CWi

[18] *Convolution arithmetic tutorial.* Theano. http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

[19] Haihan Lan. *The Softmax Function, Neural Net Outputs as Probabilities, and Ensemble Classifiers.* https://towardsdatascience.com/the-softmax-function-neural-net-outputs-as-probabilities-and-ensemble-classifiers-9bd94d75932

[20] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection.* ArXiv:1506.02640, 2015. http://arxiv.org/abs/1506.02640

[21] Joseph Redmon, Ali Farhadi. *YOLO9000: Better, Faster, Stronger.* ArXiv:1612.08242, 2016. http://arxiv.org/abs/1612.08242

[22] Joseph Redmon, Ali Farhadi. *YOLOv3: An Incremental Improvement.* ArXiv:1804.02767, 2018. http://arxiv.org/abs/1804.02767

[23] Maneesh Apte, Simar Mangat, Priyanka Sekhar *http://cs231n.stanford.edu/reports/2017/pdfs/135.pdf.* Stanford University

[24] Heng Qi, Wu Liu and, Liang Liu. *An efficient deep learning hashing neural network for mobile visual search.* ArXiv:1710.07750, 2017. https://arxiv.org/abs/1710.07750

[25] Yoojin Choi, Mostafa El-Khamy, Jungwon Lee *Universal Deep Neural Network Compression.* ArXiv:1802.02271, 2018. http://arxiv.org/abs/1802.02271

[26] Franco Manessi, Alessandro Rozza, Simone Bianco, Paolo Napoletano, Raimondo Schettini. *Automated Pruning for Deep Neural Network Compression.* ArXiv:1712.01721, 2017. https://arxiv.org/abs/1712.01721

[27] Nicholas Frosst, Geoffrey E. Hinton. *Distilling a Neural Network Into a Soft Decision Tree.* ArXiv:1712.01721, 2017. https://arxiv.org/abs/1711.09784

[28] Irsoy, Ozan and Yildiz, Olcay Taner and Alpaydin, Ethem. *Soft Decision Trees.* http://www.cs.cornell.edu/ oirsoy/files/icpr21.pdf

[29] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* ArXiv:1704.04861, 2017. https://arxiv.org/abs/1704.04861

[30] Michael Bernico *Deep Learning Quick Reference: Useful hacks for training and optimizing deep neural networks with TensorFlow and Keras. Chapter 7.* Packt Publishing Ltd, 2018

[31] Anurag Bhardwaj, Wei Di, Jianing Wei *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling* 2018

[32] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, Alexander C. Berg. *SSD: Single Shot MultiBox Detector.* ArXiv:1512.02325, 2015. https://arxiv.org/abs/1512.02325

[33] Kasthurirangan Gopalakrishnan, S.K. Khaitan, Ankit Agrawal, Alok Choudhary. *Deep Convolutional Neural Networks with transfer learning for computer vision-based data-driven pavement distress detection* https://www.researchgate.net/publication/319952138/download

[34] Karen Simonyan, Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition.* ArXiv:1409.1556, 2014. http://arxiv.org/abs/1409.1556

[35] Christian Szegedy, Scott E. Reed, Dumitru Erhan, Dragomir Anguelov. *Scalable, High-Quality Object Detection.* ArXiv:1412.1441, 2014. http://arxiv.org/abs/1412.1441

[36] Shane Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs.* 2012. [p13,14]

[37] Lei Wang, Jianfeng Zhan, Wanling Gao, Rui Ren, Xiwen He, Chunjie Luo, Gang Lu, Jingwei Li *BOPS, Not FLOPS! A New Metric, Measuring Tool, and Roofline Performance Model For Datacenter Computing* ArXiv:1801.09212, 2018. http://arxiv.org/abs/1801.09212

[38] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *CIFAR-10 dataset: 60000 32x32 colour images in 10 classes* https://www.cs.toronto.edu/ kriz/cifar.html

[39] *Numpy arrays lecture..* ArXiv:1412.1441, 2014. http://www.scipy-lectures.org/intro/numpy/numpy.html

[40] Image manipulation library used in Python.https://pypi.org/project/image-utils/

[41] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu. *scikit-image: image processing in Python.* https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4081273/

[42] Medvidovic, Nenad and Rosenblum, David S. and Redmiles, David F. and Robbins, Jason E. *Modeling Software Architectures in the Unified Modeling Language*. ACM Trans. Softw. Eng. Methodol, 2002

[43] Grady Booch, James Rumbaugh, Ivar Jacobson *Unified Modeling Language User Guide*. Addison Wesley, 1998.

[44] Repositorio github con multitud de modelos de Redes Neuronales. https://github.com/tensorflow/models

[45] M. A. Kashem and V. Ganapathy and G. B. Jasmon and M. I. Buhari `DRPT2000. International Conference on Electric Utility Deregulation and Restructuring and Power Technologies. Proceedings (Cat. No.00EX382)` *A novel method for loss minimization in distribution networks* Pages 251-256

Appendices

Appendix A. **Main Activity showing the video link from the Aircraft.**

The result of the view showing the video obtained from the drone is the following:

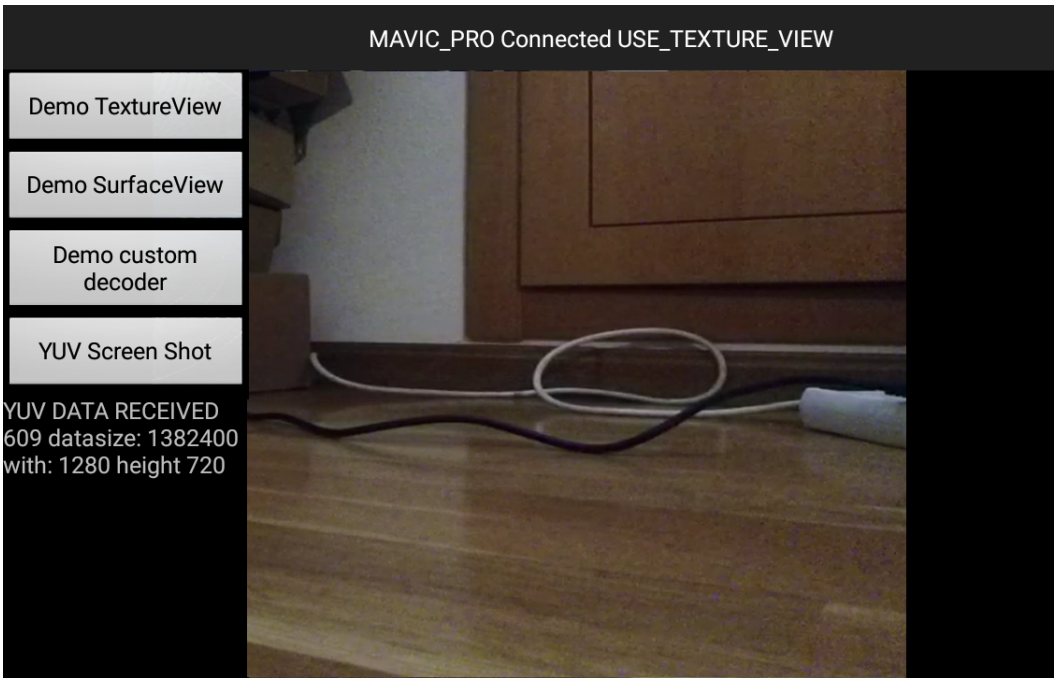


Figure A.1: Video obtained from the drone DJI Mavic Pro

Appendix B. **Neural Network configuration**

The configuration file used in the Neural Network training process is the following:

```
model {
  ssd {
    num_classes: 3
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
    box_predictor {
      convolutional_box_predictor {
        min_depth: 0
        max_depth: 0
        num_layers_before_predictor: 0
        use_dropout: false
        dropout_keep_probability: 0.8
        kernel_size: 1
        box_code_size: 4
        apply_sigmoid_to_scores: false
        conv_hyperparams {
          activation: RELU_6,
          regularizer {
            l2_regularizer {
              weight: 0.00004
            }
          }
          initializer {
            truncated_normal_initializer {
              stddev: 0.03
              mean: 0.0
            }
          }
          batch_norm {
            train: true,
            scale: true,
            center: true,
            decay: 0.9997,
            epsilon: 0.001,
          }
        }
      }
    }
    feature_extractor {
      type: 'ssd_mobilenet_v1'
      min_depth: 16
      depth_multiplier: 1.0
      conv_hyperparams {
        activation: RELU_6,
        regularizer {
          l2_regularizer {
            weight: 0.00004
          }
        }
```

```
        initializer {
          truncated_normal_initializer {
            stddev: 0.03
            mean: 0.0
          }
        }
        batch_norm {
          train: true,
          scale: true,
          center: true,
          decay: 0.9997,
          epsilon: 0.001,
        }
      }
    }
    loss {
      weighted_sigmoid {
      }
      weighted_smooth_l1 {
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
  batch_size: 4
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
        exponential_decay_learning_rate {
          initial_learning_rate: 0.004
          decay_steps: 800720
          decay_factor: 0.95
        }
      }
      momentum_optimizer_value: 0.9
      decay: 0.9
      epsilon: 1.0
    }
  }
  fine_tune_checkpoint: "ssd_mobilenet_v1_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  num_steps: 60000
  data_augmentation_options {
```

```
    random_horizontal_flip {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "data/train.record"
  }
  label_map_path: "data/object-detection.pbtxt"
}

eval_config: {
  num_examples: 584
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "data/test.record"
  }
  label_map_path: "data/object-detection.pbtxt"
  shuffle: false
  num_readers: 1
}
```

Listing B.1: Neural Network configuration file

Appendix C. **DJIDetectorActivity class code**

The final code of the DJIDetectorActivity class:

```
public class DJIDetectorActivity extends Activity {
    Button changeDetector;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //...Initialization of the components...//
        new Thread(() -> {
                while(true) {
                    Thread.sleep(33);
                    processImage();
                    refreshView();
                }
        }).start();
    }

public static void onFrameReady(byte[] yuvBytes){
    int[] rgbBytes = convertYUV420_NV21toRGB8888(yuvbytes, width,
    height);
    bmp = createBitmap(rgbBytes, width, hight, Bitmap.ARGB_8888);
}

public void refreshView(){
    runOnUiThread(new Runnable () -> {
        if(bmp != null) ivDetector.setImageBitmap(bmp);
        ivDetector.postInvalidate();
    };);
}

protected void processImage() {
 new Runnable() {
 @Override
 public void run() {
  List<Recognition> results = detector.recognizeImage(bmp);
  copyBitmap = Bitmap.createBitmap(bmp);
  final Canvas canvas = new Canvas(copyBitmap);
  final Paint paint = new Paint();
  paint.setColor(Color.RED);
  paint.setStyle(Style.STROKE);
  paint.setStrokeWidth(2.0f);

  float minimumConfidence = 0;
  switch (MODE) {
    case SSD_MOBILENET:
    minimumConfidence = MINIMUM_CONFIDENCE_SSD_MOBILENET;
    break;
    case YOLO:
    minimumConfidence = MINIMUM_CONFIDENCE_YOLO;
    break;
  }

  List<Recognition> mappedRecognitions = new LinkedList<>();

  for (final Classifier.Recognition result : results) {
```

```
53      final RectF location = result.getLocation();
54      if (result.getConfidence() >= minimumConfidence) {
55      canvas.drawRect(location, paint);
56      result.setLocation(location);
57      mappedRecognitions.add(result);
58      }
59    }
60    ivOverlay.paint(canvas);
61   });
62 }
```

Listing C.1: Code of the class DJIDetectorActivity

Appendix D. **Code of the class CommandAircraft.java**

The code of the class CommandAircraft is the following:

```
public class CommandAircraft {
  private float pitch = 0f;
  private float roll = 0f;
  private float yaw = 0f;
  private float throttle = 0f;
  private int image_width = 1280;
  private int image_height = 720;

  private Timer sendVirtualStickDataTimer;
  private SendVirtualStickDataTask sendVirtualStickDataTask;
  private FlightController fContr;

  public void init(){
    fController = ((Aircraft) DJISDKManager.getInstance()
      .getProduct()).getFlightController();

    sendVirtualStickDataTask = new SendVirtualStickDataTask();
    sendVirtualStickDataTimer = new Timer();
    sendVirtualStickDataTimer.schedule(sendVirtualStickDataTask, 200);
    fContr.setVerticalControlMode(VerticalControlMode.VELOCITY);
    fContr.setYawControlMode(YawControlMode.ANGULAR_VELOCITY);
    fContr.setRollPitchControlMode(RollPitchControlMode.VELOCITY);
  }

  public void setCommand(RectF location){
    float boxWidth = location.right - location.left;
    pitch = (image_width/3) - boxWidth;

    float boxCenter = location.left + (boxWidth/2);
    yaw = boxCenter - (image_width/2);
  }


  private class SendVirtualStickDataTask extends TimerTask {
    @Override
    public void run() {
      if ( fContr != null) {
        fContr.sendVirtualStickFlightControlData(
          new FlightControlData(pitch,roll,yaw,hrottle);
      }
    }
  }
}
```

Listing D.1: Code of the class Command Airctraft

Appendix E. **Final application view**

The final application view displaying a detection of the drone DJI Mavic PRO looks as follows:

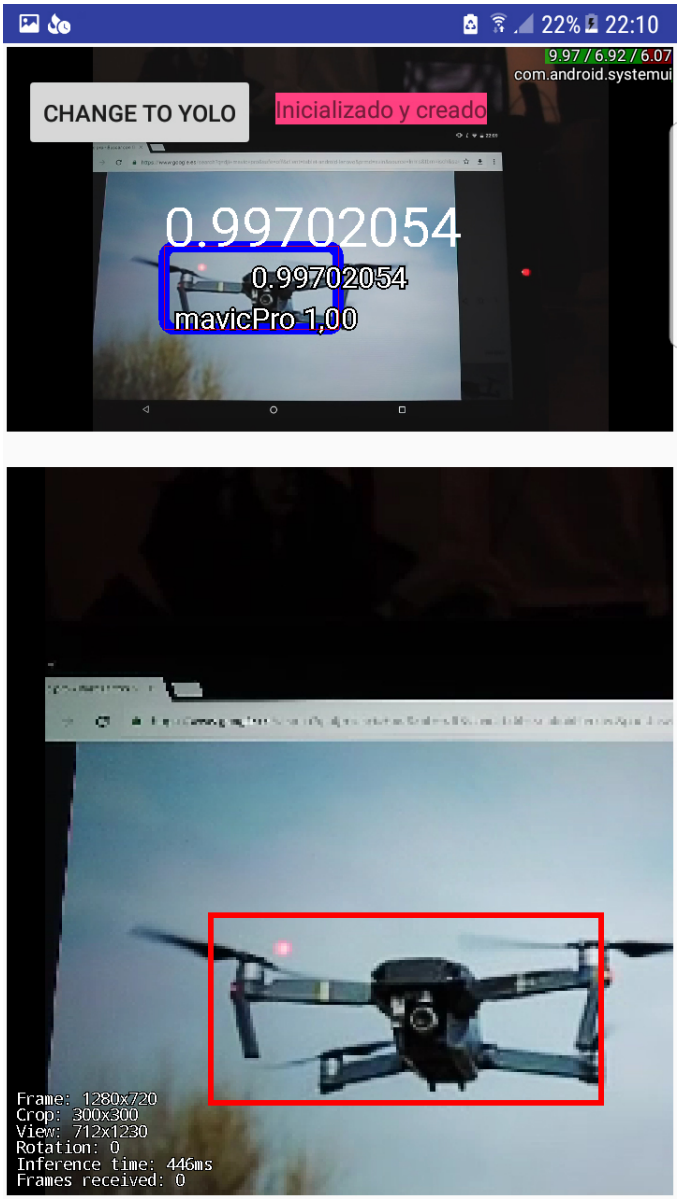


Figure E.1: The application running on a mobile device displaying a detection.